VISIT…

INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE

Logic and Integer Programming

H. P. Williams

Springer

Logic and Integer Programming

INT. SERIES IN OPERATIONS RESEARCH & MANAGEMENT SCIENCE

Series Editor: Frederick S. Hillier, Stanford University

Special Editorial Consultant: Camille C. Price, Stephen F. Austin State University

Titles with an asterisk (*) were recommended by Dr. Price

Saaty & Vargas/ *DECISION MAKING WITH THE ANALYTIC NETWORK PROCESS: Economic, Political, Social & Technological Applications w. Benefits, Opportunities, Costs & Risks*
Yu/*TECHNOLOGY PORTFOLIO PLANNING AND MANAGEMENT: Practical Concepts and Tools*
Kandiller/ *PRINCIPLES OF MATHEMATICS IN OPERATIONS RESEARCH*
Lee & Lee/ *BUILDING SUPPLY CHAIN EXCELLENCE IN EMERGING ECONOMIES*
Weintraub/ *MANAGEMENT OF NATURAL RESOURCES: A Handbook of Operations Research Models, Algorithms, and Implementations*
Hooker/ *INTEGRATED METHODS FOR OPTIMIZATION*
Dawande et al/ *THROUGHPUT OPTIMIZATION IN ROBOTIC CELLS*
Friesz/ *NETWORK SCIENCE, NONLINEAR SCIENCE and INFRASTRUCTURE SYSTEMS*
Cai, Sha & Wong/ *TIME-VARYING NETWORK OPTIMIZATION*
Mamon & Elliott/ *HIDDEN MARKOV MODELS IN FINANCE*
del Castillo/ *PROCESS OPTIMIZATION: A Statistical Approach*
Józefowska/*JUST-IN-TIME SCHEDULING: Models & Algorithms for Computer & Manufacturing Systems*
Yu, Wang & Lai/ *FOREIGN-EXCHANGE-RATE FORECASTING WITH ARTIFICIAL NEURAL NETWORKS*
Beyer et al/ *MARKOVIAN DEMAND INVENTORY MODELS*
Shi & Olafsson/ *NESTED PARTITIONS OPTIMIZATION: Methodology and Applications*
Samaniego/ *SYSTEM SIGNATURES AND THEIR APPLICATIONS IN ENGINEERING RELIABILITY*
Kleijnen/ *DESIGN AND ANALYSIS OF SIMULATION EXPERIMENTS*
Førsund/ *HYDROPOWER ECONOMICS*
Kogan & Tapiero/ *SUPPLY CHAIN GAMES: Operations Management and Risk Valuation*
Vanderbei/ *LINEAR PROGRAMMING: Foundations & Extensions, 3rd Edition*
Chhajed & Lowe/*BUILDING INTUITION: Insights from Basic Operations Mgmt. Models and Principles*
Luenberger & Ye/*LINEAR AND NONLINEAR PROGRAMMING, 3rd Edition*
Drew et al/ *COMPUTATIONAL PROBABILITY: Algorithms and Applications in the Mathematical Sciences**
Chinneck/ *FEASIBILITY AND INFEASIBILITY IN OPTIMIZATION: Algorithms and Computation Methods*
Tang, Teo & Wei/ *SUPPLY CHAIN ANALYSIS: A Handbook on the Interaction of Information, System and Optimization*
Ozcan/ *HEALTH CARE BENCHMARKING AND PERFORMANCE EVALUATION: An Assessment using Data Envelopment Analysis (DEA)*
Wierenga/*HANDBOOK OF MARKETING DECISION MODELS*
Agrawal & Smith/ *RETAIL SUPPLY CHAIN MANAGEMENT: Quantitative Models and Empirical Studies*
Brill/ *LEVEL CROSSING METHODS IN STOCHASTIC MODELS*
Zsidisin & Ritchie/ *SUPPLY CHAIN RISK: A Handbook of Assessment, Management & Performance*
Matsui/ *MANUFACTURING AND SERVICE ENTERPRISE WITH RISKS: A Stochastic Management Approach*
Zhu/*QUANTITATIVE MODELS FOR PERFORMANCE EVALUATION AND BENCHMARKING: Data Envelopment Analysis with Spreadsheets*
Kubiak/ *PROPORTIONAL OPTIMIZATION AND FAIRNESS**
Bier & Azaiez/ *GAME THEORETIC RISK ANALYSIS OF SECURITY THREATS**
Cox/*RISK ANALYSIS OF COMPLEX AND UNCERTAIN SYSTEMS*

A list of the early publications in the series is found at the end of the book

H. Paul Williams

Logic and Integer Programming

Springer

H. Paul Williams
London School of Economics
Department of Operations Research
Houghton St.
London
United Kingdom WC2A 2AE
h.p.williams@lse.ac.uk

ISBN 978-0-387-92279-9 e-ISBN 978-0-387-92280-5
DOI 10.1007/978-0-387-92280-5
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2008943036

© Springer Science+Business Media, LLC 2009
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Isobel, Imogen and Edward

Preface

This book combines two related topics which are usually covered in separate texts, namely logic and integer programming (discrete optimisation). These two subjects have close connections and each is applicable to the other.

Much of the work which makes up this book is due to others as well as the present author, in particular John Hooker and the late Robert Jeroslow. John Hooker has written a number of comprehensive books on the subjects which are referenced in the appropriate places.

This book is shorter than these and intended to give a readable and succinct coverage of the subjects. It is intended for students (and practitioners and academics) in operational research, computer science and mathematics.

Logic, by definition, involves a high level of abstraction as it is intended to separate interpretations of logical systems from the structure of those systems. This can make for difficulties in comprehension as understanding is usually motivated by using concrete examples. In this book we have placed the emphasis on comprehension by motivating the discussion by interpretations while pointing out when we are doing this.

We avoid the usual format of definitions, theorems and proofs by introducing concepts and results within the text (usually italicised) by examples. References are given at the end of each chapter to further (often more mathematical) papers and texts on topics.

Optimisation problems (in practice linear and integer programming) are not usually presented in a logical context. However, doing this enables one to give their statement, as models, a greater precision. This often throws greater insight into their mathematical nature and properties. In addition logic is a valuable tool for modelling, and sometimes solving such models.

Chapter 1 gives a basic introduction to logic and its aims as well as explaining the propositional and predicate calculus. Chapter 2 explains linear and integer programming (LP and IP) using the machinery of logic. It also explains the fundamental structural and mathematical properties of these types of models. The main methods of solving IP models are described. The most common types of IP model restrict the variables to 0 or 1 (with the obvious logical false/true interpretation). The major areas of practical application are explained. Also a section is devoted to the attempt to distinguish between computationally 'easy' classes of problem

and 'difficult' classes of problem under the subject of 'computational complexity'. Chapter 3 applies logic to the formulation of IP models using the methods explained in Chapter 1. There are often 'good' and 'bad' ways of modelling IPs (from both the explanatory and ultimately solvability points of view). Also the deeper mathematical concepts involved (e.g. 'convexification') are covered here. Chapter 4 covers the fundamental problem of computational logic, namely the 'satisfiability problem'. This problem lies at the heart of most of what is covered in this book. Methods of solving this problem through both logic and integer programming are given and their connections described. Applications are given in a number of diverse fields. It is also shown how IP models can be expressed as satisfiability problems and solved as such.

Readers familiar with either logic or IP or with both may be able to skip appropriate sections. However, it is expected that all chapters will contain material not familiar to everybody and they are recommended to read the book in its entirety. If it is desired to pursue a particular topic in greater depth then it is hoped that the references at the end of each chapter will provide a route into the topic. The references are therefore intended for this purpose. I am aware that there are many research papers not referenced, covering significant work. I apologise, in advance, to any authors who might feel slighted, but the intention of the book is to explain and provide pursuable introductions to topics rather than comprehensively cover all work done. Exercises are given at the end of each chapter. These are intended to reinforce, and sometimes expand on, the material in the chapter. As with many mathematical subjects attempting the exercises is often the best way of understanding.

An obvious question, which a reader might sensibly ask, is "what is prescribed as the best method of solving a practical problem". No definitive answer is given in this book. Different problems demand different approaches. However, what is needed is a thorough knowledge of basic logic for the purpose of clarifying and modelling discrete optimisation problems. Also needed is a knowledge of how linear and integer programming can be used to solve the resultant models, often enhanced by using logical methods. It is hoped that the material in this book will help in both these areas.

I am indebted to a number of friends and colleagues for reading and commenting on sections of this book. In particular I would like to mention Gautam Appa, Nikos Argyris, John Hooker and John Wilson.

Also I would like to acknowledge the help I have received from Carol Hewlett, Nikos Argyris and Mara Airoldi in the use of the software Scientific Workplace, Latex and IPE in the preparation of this book.

Finally I would like to acknowledge the help which resulted from Leverhulme Research Fellowship RF&G/10185 and EPSRC Overseas Travel Grant EP/C530578/1 in preparing this book.

Winchester, UK H. Paul Williams

Contents

1 An Introduction to Logic 1
1.1 The Purpose of Logic: Philosophical: Computational 1
1.2 Logical Inference and Consistency 2
1.3 The Propositional Calculus 3
1.3.1 Connectives and Truth Tables 3
1.3.2 Equivalent Statements 5
1.3.3 Disjunctive and Conjunctive Normal Forms 6
1.3.4 Complete Sets of Connectives 10
1.3.5 The Calculus of Indications 11
1.3.6 Venn Diagrams 13
1.4 The Predicate Calculus 14
1.4.1 The Use of Quantifiers 15
1.4.2 Prenex Normal Form 16
1.5 Decidable Fragments of Mathematics 18
1.5.1 The Theory of Dense Linear Order 18
1.5.2 Arithmetic Without Multiplication 20
1.6 References and Further Work 22
1.7 Exercises 22

2 Integer Programming 25
2.1 Linear Programming 25
2.1.1 The Dual of an LP Model 28
2.1.2 A Geometrical Representation of a Linear Programme . . 30
2.2 Integer Programming 35
2.2.1 The Branch-and-Bound Algorithm 38
2.2.2 The Convex Hull of an IP 43
2.3 The Use of 0–1 Variables 49
2.3.1 Expressing General Integer Variables as 0–1 Variables . . 49
2.3.2 Yes/No Decisions 50
2.3.3 The Facility Location Problem 50
2.3.4 Logical Decisions 51
2.3.5 Products of 0–1 Variables 53

2.3.6 Set-Covering, Packing and Partitioning Problems 53
2.3.7 Non-linear Problems 55
2.3.8 The Knapsack Problem 58
2.3.9 The Travelling Salesman Problem 58
2.3.10 Other Problems 60
2.4 Computational Complexity 60
2.4.1 Problem Classes and Instances 60
2.4.2 Computer Architectures and Data Structures 61
2.4.3 Polynomial and Exponential Algorithms 62
2.4.4 Non-deterministic Algorithms and Polynomial Reducibility 64
2.4.5 Feasibility Versus Optimisation Problems 65
2.4.6 Other Complexity Concepts 66
2.5 References and Further Work 66
2.6 Exercises 68

3 Modelling in Logic for Integer Programming 71
3.1 Logic Connectives and IP Constraints 71
3.2 Disjunctive Programming 75
3.2.1 A Geometrical Representation 75
3.2.2 Mixed IP Representability 78
3.3 Alternative Representations and Tightness of Constraints 84
3.3.1 Disjunctive Versus Conjunctive Normal Form 86
3.3.2 The Dual of a Disjunctive Programme 89
3.4 Convexification of an IP Model 91
3.4.1 Splitting Variables 92
3.5 Modelling Languages Based On Logic 96
3.5.1 Algebraic Languages 96
3.5.2 The 'Greater Than or Equal' Predicate 98
3.6 References and Further Work 102
3.7 Exercises 102

4 The Satisfiability Problem and Its Extensions 105
4.1 Resolution and Absorption 106
4.2 The Davis–Putnam–Loveland (DPL) Procedure 109
4.3 Representation as an Integer Programme 109
4.4 The Relationship Between Resolution and Cutting Planes 111
4.5 The Maximum Satisfiability Problem 113
4.6 Simplest Equivalent Logical Statement 116
4.7 Horn Clauses: Simple Satisfiability Problems 119
4.8 Constraint Logic Programming 123
4.8.1 Modelling in CLP 124
4.8.2 Solving CLP Models 126
4.8.3 Hybrid CLP and IP systems 127

4.9 Solving Integer Programmes as Satisfiability Problems 129
4.10 Applications 134
4.10.1 Electrical Circuit Design Using Switches 134
4.10.2 Logical Net Design Using Gates 135
4.10.3 The Logical Analysis of Data (LAD) 137
4.10.4 Chemical-processing networks 139
4.10.5 Other Applications 140
4.11 References and Further Work 141
4.12 Exercises 142

References 145

Index 151

Chapter 1
An Introduction to Logic

1.1 The Purpose of Logic: Philosophical: Computational

Traditionally logic has been concerned with the *form* of statements as opposed to their *content.* The aim is to produce a system in which it is possible to deduce *true* statements from other 'true' statements independently of what one is talking about, i.e. independently of the *interpretation* of the statements. The usual approach is to start with a set of *axioms* and *rules of deduction* and produce statements which are true if the axioms are true. Such statements are said to be *analytically* true, as opposed to statements which are regarded as true on the basis of experimental evidence in, e.g. sciences such as physics, biology, psychology, etc. A major philosophical aim in the late 19th and early 20th centuries was to use logic to put mathematics on a rigorous footing. It was then hoped that mathematical statements could be regarded as having an absolute truth in this sense. The pattern which it was hoped to follow was that done by Euclid for geometry which was very successful. Russell and Whitehead devoted an enormous amount of time to trying to axiomatise mathematics. They regarded mathematics as simply an extension of logic. Unfortunately their efforts were largely in vain as a result of two discoveries, mainly due to Gödel.

First it was shown that it was impossible to prove the *consistency* of the axiom system without using methods which went beyond the mathematical system itself. A *meta system* was needed which was richer than the original system. If a system is not consistent then, in any worthwhile form of logic, one can prove any statement (and its negation) rendering the system vacuous.

Second it was shown that, however, many axioms one had there would always be 'true' statements which could not be proved (unless one added new axioms in which case further 'true' statements could be found).

The confusion between 'truth' and provability was responsible for the difficulties in formalising mathematics. The former is a less well-defined concept than the latter. As a result of the failure of Russell and Whitehead's agenda, attention turned to less ambitious aims of only putting 'parts' (*fragments)* of mathematics on a rigorous footing (e.g. restricted forms of set theory). Some of these fragments have proved useful in a computational setting (e.g. arithmetic without multiplication and the Theory of dense linear order). The purpose of this book is to use logic for

H.P. Williams, *Logic and Integer Programming,* International Series in Operations Research & Management Science 130, DOI 10.1007/978-0-387-92280-5_1,
© Springer Science+Business Media, LLC 2009

modelling and computational purposes. The methods of logic can be used to solve less ambitious problems than the formalisation of all mathematics. The problems which we use it for, in this book, arise mainly in operational research and computer science. They are often (but not always) *optimisation* problems.

However, the efficiency of the computational methods used depends first on how the models are built and second the methods used, some of which rely on logic. We will mainly be concerned with *discrete optimisation,* i.e. where some quantities are restricted to taking discrete values, as opposed to a continuum of (say) the real or rational numbers. It is not always obvious, with any one problem, to what extent one uses logic or to what extent one uses more traditional methods. There is great advantage, however, in being able to move between the two and recognise the relationships between them. In this sense discrete optimisation (usually known as *integer programming*) and logic are symbiotic.

1.2 Logical Inference and Consistency

As already stated logic is concerned with *deducing* statements from other statements. This is the process known as *inference.* We formalise this in Sects. 1.3 and 1.4 where we represent statements symbolically and show how to manipulate them. Sometimes this is referred to as *symbolic logic.*

For example

$$\textit{Is it valid to make the following inference ?} \tag{1.1}$$

Glasgow is in Scotland and England and Scotland are part of Britain

Manchester is in England or Scotland (1.2)

Therefore

$$\textit{Manchester and Glasgow are both in Britain} \tag{1.3}$$

Note that (1.1) is a question which depends on the structure of the sentences and rules of deduction to be used, i.e. it is not a question which needs a knowledge of geography to answer. Also it is a question **about** the system of statements we are working with, not a statement **within** the system we are dealing with (which (1.2) and (1.3) both are). It is said to be a statement within the *meta system* as opposed to the system. The meta system is the language for talking about the system. Attempts to encompass a meta system within the language of the system is what led to a lot of the paradoxes (e.g. Russell's paradox) in mathematics. In order to distinguish meta statements from statements we will use the symbol '$\Rightarrow$' to represent inference. By saying $A \Longrightarrow B$ we are meaning that from the statement (or set of statements) A we can infer the statement B. The set of *premises* in (1.2) can be represented by A and the *conclusion* (1.3) represented by B. *Consistency* is concerned with whether a system can ever produce a contradiction, i.e. can we infer both a statement and its negation? Again consistency is a meta concept. It is a property of a system not

a statement within a system. Consistency and inference are closely related to each other. A set of statements is only consistent if we cannot infer a contradiction from them. A conclusion can only be inferred from a set of premises if the negation of the conclusion is inconsistent with the premises.

Another important property of a logical system consisting of statements, axioms and rules of deduction is *completeness.* A system is complete if every statement in the system can either be proved to be true or false.

As a result of Gödel's work it was found that a system rich enough to encompass full arithmetic could not be shown to be either consistent or have an axiom system which makes it complete. However, the system that we will be using most, in this book, namely the *propositional calculus* (also known as Boolean algebra) is both consistent and can be given an axiom system which makes it complete. Also the richer system, known as the *Predicate Calculus* is consistent and has a complete axiomatisation.

In this book we will not be using an axiomatic treatment. As we are more concerned with applying logic and integer programming for practical (as opposed to philosophical, important though this is) purposes we will not be concerned with proving the above properties or otherwise. We will use more intuitive methods when necessary.

There is a way of viewing inference which connects with another theme of this book. That is *optimisation.* Linear and integer programming (introduced in Chapter 2) are concerned with *maximising* or *minimising* linear expressions subject to linear constraints (in real or integer numbers, respectively). As such they are attempting to find the *strongest* linear constraints which can be inferred from the original constraints. Therefore, optimisation can be viewed as a method of inference. When a logical verification problem is cast into the form of an optimisation problem it can be solved as such. This is done in Chapter 3. Alternatively integer programming models can be cast as logical inference problems as done in Chapter 4.

A mathematical or logical system is said to be *decidable* if there exists a 'mechanical' procedure for deciding the truth or falsity of a statement in the system. Such a procedure is known as a *decision procedure* or an *algorithm*. Again the methods of Gödel and others demonstrate that full arithmetic is not decidable. Some 'smaller' systems are also undecidable. However, the propositional calculus, which we are mainly concerned with in this book, is decidable. The predicate calculus is not decidable although there are decision procedures for always deciding if a statement is false (but not if a statement is true). Also appending elements to the predicate calculus can produce decidable systems as described in Sect. 1.5.

1.3 The Propositional Calculus

1.3.1 Connectives and Truth Tables

This system is sometimes called '*Boolean algebra*'. It combines *atomic statements,* which can take truth values **true** (T) or **false** (F) **,** into *compound statements* whose truth depends, by the rules of this calculus, on the truth values of the component

atomic statements. We will represent atomic statements by *literals* A, B, X_1, X_2, etc. and their negations by $\overline{A}, \overline{B}, \overline{X}_1, \overline{X_2}$, etc. (the former set of literals will be said to have 'positive sign' and the latter set of literals 'negative sign'). Both the individual letters and their negations are known as literals. The atomic statements and compound statements are combined by *connectives* to produce more compound statements whose truth value depends on the truth values of the component statements. The connectives which we will initially consider are '$\vee$' ('or') , '$\cdot$' ('and') , '$-$' ('not') , '$\longrightarrow$' ('implies') and '$\longleftrightarrow$' ('if and only if'). The effects of these are given in the **truth table** below.

Table 1.1 The Truth Table for Common Connectives

A	B	$A \vee B$	$A \cdot B$	$\overline{A}$	$A \longrightarrow B$	$A \longleftrightarrow B$
T	T	T	T	F	T	T
T	F	T	F	F	F	F
F	T	T	F	T	T	F
F	F	F	F	T	T	T

Table 1.1 can be taken as the *definition* of these connectives. When the truth table is applied to a compound statement we have a *decision* procedure for determining the truth or otherwise of the statement. We use this as an alternative to the axiomatic approach. However, there are often more efficient decision procedures (e.g. integer programming) for particular problems which are discussed in subsequent chapters of this book. Some discussion should be given to the nature of the connectives defined in Table 1.1. Since these are the definitions they need not, necessarily, totally conform to common usage. '$\vee$' is sometimes called the 'inclusive or' since it is also true when both the component statements are true (as opposed to the 'exclusive or' which is not true if both components are true). '$\longrightarrow$' does not represent implication in any causal sense. In particular it is defined as true when neither component statement is true, when in practice it might be thought to be inapplicable. It is only false when the first statement is true and the second false. The connective '$\longleftrightarrow$' is sometimes called 'equivalent to' since the two component statements must have the same truth value for it to be true. It must be emphasised again that these connectives are within the system. For example '$\Longrightarrow$' is a meta relation about statements in the system although it obviously has a semantic correspondence with '$\longrightarrow$'. Similarly '$\longleftrightarrow$' should not be confused with the meta relation '$\equiv$' which means two statements within the system are equivalent to each other.

Apart from '$-$' (which can be regarded as a connective applying to one statement, which is written above the statement) all the other connectives in Table 1.1 connect two component statements. Obviously there are 16 (2^4) possible ways of assigning the two truth values T and F to the four rows of a truth table connecting two statements. One of these is to assign T to each row indicating what is known as a '*tautology*' (always true). Another is to assign F to each row indicating what is known as a '*contradiction*' (always false). There are, however, a number of other connectives (9 in total) which we have not defined in Table 1.1 but which may be useful in some circumstances.

1.3.2 Equivalent Statements

It is easy to verify, by means of truth tables (and is left as Exercise 1.7.2), that the following equivalences hold:

$$A \cdot B \equiv B \cdot A \tag{1.4}$$

$$A \vee B \equiv B \vee A \tag{1.5}$$

$$A \cdot (B \vee C) \equiv (A \cdot B) \vee (A \cdot C) \tag{1.6}$$

$$A \vee (B.C) \equiv (A \vee B) \cdot (A \vee C) \tag{1.7}$$

$$A \cdot B \equiv\sim (\overline{A} \vee \overline{B}) \tag{1.8}$$

$$A \vee B \equiv\sim (\overline{A} \cdot \overline{B}) \tag{1.9}$$

$$A \longrightarrow B \equiv \overline{A} \vee B \tag{1.10}$$

$$A \longleftrightarrow B \equiv (A \longrightarrow B) \cdot (B \longrightarrow A) \tag{1.11}$$

$$A \longrightarrow B \equiv \overline{B} \longrightarrow \overline{A} \tag{1.12}$$

$$A \cdot \overline{A} \equiv F \tag{1.13}$$

$$A \vee \overline{A} \equiv T \tag{1.14}$$

$$\overline{\overline{A}} \equiv A \tag{1.15}$$

$$A \cdot T \equiv A \tag{1.16}$$

$$A \cdot F \equiv F \tag{1.17}$$

$$A \vee T \equiv T \tag{1.18}$$

$$A \vee F \equiv A \tag{1.19}$$

$$A \cdot A \equiv A \tag{1.20}$$

$$A \vee A \equiv A \tag{1.21}$$

Note the use of the *meta symbol* for the equivalence between statements.

Equations (1.4) and (1.5) are known as the *commutative laws* and (1.6) and (1.7) as the *distributive laws*. (They are analogous to similar laws for addition and multiplication in arithmetic, although if '$\vee$' is analogous to addition and '$\cdot$' to multiplication (1.7) does not apply.) Equations (1.8) and (1.9) are known as *De Morgan's laws*. They demonstrate a symmetry between the '$\vee$' and '$\cdot$' connectives. These laws enable one to manipulate expressions into equivalent expressions and standardise them into *normal forms* as described below. Repeated application of De Morgan's laws to a statement, in order to negate it, changes all the '$\vee$' connectives to '$\cdot$' and vice versa and negates the unnegated literals and unnegates the negated ones. The resultant statement is said to be the *logical dual* of the original and vice versa.

Equation (1.12) is known as the *contrapositive*. Intuitively it makes sense. B is True only if A is true. Therefore it B is not true then A cannot be true. This transformation is very useful in certain integer programming applications as described in Chapters 3 and 4.

Compound statements that are always true for any truth settings of the atomic statements (such as (1.14)) are known as *tautologies.* In contrast compound statements that are always false (such as (1.13)) , for any truth settings of the atomic statements, are known as *contradictions.*

In order to aid readability it is convenient to regard the '$\cdot$' connective as more binding than the other connectives. This enables one to dispense with the brackets in the right-hand expression in (1.6) and the left-hand expression in (1.7). Note that in many texts the symbol '$\wedge$' is used instead of '$\cdot$' and negation of a statement A is indicated by $\sim A$ instead of $\overline{A}$. We use our notation to aid readability but sometimes use $\sim$ instead of $-$ when we wish to negate a compound statement instead of a literal and wish to avoid long cumbersome bars across the top of an expression.

1.3.3 Disjunctive and Conjunctive Normal Forms

Let us consider a truth table for a statement S with n component statements referred to as $A_1, A_2, \ldots, A_n$. We represent this in Table 1.2.

Table 1.2 A General Truth Table

A_1	A_2		A_n	S
T	T	...	T	–
T	T		F	–
	⋮		⋮	⋮
F	F	...	F	–

Let S have the value T for rows $i_1, i_2, \ldots, i_r$ and F for all the other rows and the entries for $A_1, A_2, \ldots, A_n$ in row i_j are T for columns $k_{i_j,i_j^1}, k_{i_j,i_j^2}, \ldots, k_{i_j,i_j^{p_{i_j}}}$ and F for columns $k_{i_j,i_j^{p_{i_j}+1}}, k_{i_j,i_j^{p_{i_j}+2}}, \ldots, k_{i_j,i_j^n}$. Then we can represent row i_j by the statement

$$A_{k_{i_j,i_j^1}} \cdot A_{k_{i_j,i_j^2}} \cdot \cdots \cdot A_{k_{i_j,i_j^{p_{i_j}}}} \cdot \overline{A_{k_{i_j,i_j^{p_{i_j}+1}}}} \cdot \overline{A_{k_{i_j,i_j^{p_{i_j}+2}}}} \cdot \cdots \cdot \overline{A_{k_{i_j,i_j^n}}} \tag{1.22}$$

This is easily verified by the truth table for '$\cdot$' since, for it to be true, the atomic statements $A_{k_{i_j,l}}$ must take the values specified for this row of the truth table. Equation (1.22) is known as a *conjunctive clause* since the '$\cdot$' connective is also known as a 'conjunction'.

Since S takes the value T for each of the rows $i_1, i_2, \ldots, i_r$ S will be true if

$$\begin{aligned}
& A_{k_{i_1,i_1^1}} \cdot A_{k_{i_1,i_1^2}} \cdot \cdots \cdot A_{k_{i_1,i_1^{p_{i_1}}}} \cdot \overline{A_{k_{i_{1j},i_1^{p_{i_1}+1}}}} \cdot \overline{A_{k_{i_1,i_1^{p_{i_1}+2}}}} \cdot \cdots \cdot \overline{A_{k_{i_1,i_1^n}}} \\
\vee\ & A_{k_{i_2,i_2^1}} \cdot A_{k_{i_2,i_2^2}} \cdot \cdots \cdot A_{k_{i_2,i_2^{p_{i_2}}}} \cdot \overline{A_{k_{i_2,i_2^{p_{i_2}+1}}}} \cdot \overline{A_{k_{i_2,i_2^{p_{i_2}+2}}}} \cdot \cdots \cdot \overline{A_{k_{i_2,i_2^n}}} \\
& \qquad\qquad \vdots \\
\vee\ & A_{k_{i_r,i_r^1}} \cdot A_{k_{i_r,i_r^2}} \cdot \cdots \cdot A_{k_{i_r,i_r^{p_{i_r}}}} \cdot \overline{A_{k_{i_r,i_r^{p_{i_r}+1}}}} \cdot \overline{A_{k_{i_r,i_r^{p_{i_r}+2}}}} \cdot \cdots \cdot \overline{A_{k_{i_r,i_r^n}}}
\end{aligned} \tag{1.23}$$

i.e. if any of the conjunctive clauses is true. Statement (1.23) is known as a *disjunction* of the clauses since they are combined by the '$\vee$' connective and '$\vee$' is also known as a 'disjunction'.

Statement (1.23) is said to be in *extended disjunctive normal form* (EDNF). For a statement to be in EDNF it is made up as a disjunction of conjunctive clauses each of which contains literals (atomic statements which are negated or unnegated) for **all** atomic statements in the original statement (1.23) serves to prove that any statement in the propositional calculus can be written using the connectives '$-$', '$\vee$' and '$\cdot$'. Of course there are many other ways of writing a statement either using just these connectives or other connectives. However, EDNF also provides a standard (if sometimes uneconomical) and unique way (up to the order of the literals and clauses) of expressing any statement. The following example demonstrates its use

Example 1.1 Express statement S as defined in Table 1.3 using EDNF.

Table 1.3 A Function Defined by a Truth Table

A	B	C	S
T	T	T	F
T	T	F	F
T	F	T	T
T	F	F	F
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

Taking rows 3, 5, 6, 7 and 8 of the table gives the statement

$$A \cdot \overline{B} \cdot C \vee \overline{A} \cdot B \cdot C \vee \overline{A} \cdot B \cdot \overline{C} \vee \overline{A} \cdot \overline{B} \cdot C \vee \overline{A} \cdot \overline{B} \cdot \overline{C} \tag{1.24}$$

Often we can be content with (non-extended) *disjunctive normal form* (DNF). Here we again express a statement as a disjunction of conjunctions but do not insist that each conjunction contains literals for each atomic statement in the original statement. Such a form lacks uniqueness but will generally be more compact. Converting it into the *simplest* form is addressed in Chapter 4.

Example 1.2 demonstrates how the laws (1.4) – (1.21) can be used to convert a statement into EDNF.

Example 1.2 Convert the following statement into EDNF:

$$(A \longrightarrow B) \cdot \overline{A} \cdot C \tag{1.25}$$

Using the equivalencies given above we can successively convert (1.25) to

$$(\overline{A} \vee B) \cdot \overline{A} \cdot C \tag{1.26}$$

$$\overline{A} \cdot C \vee \overline{A} \cdot B \cdot C \tag{1.27}$$

This statement is in DNF but not in EDNF since the first clause does not contain the literal B. We can expand it to

$$\overline{A} \cdot B \cdot C \vee \overline{A} \cdot \overline{B} \cdot C \tag{1.28}$$

using equivalence (1.14). Equation (1.28) also contains the second conjunctive clause in (1.27) and is therefore the equivalent statement to (1.25) in EDNF.

There is another normal form which is widely used. This is *extended conjunctive normal form* (ECNF). It consists of a conjunction of disjunctive clauses (each containing literals for all atomic statements in the extended case). We can use De Morgan's laws to convert a statement from EDNF to ECNF and vice versa. However, there is a more straightforward way of giving a truth table representation of a statement in ECNF as well as one in EDNF. We observe that the *negation* of the statement is represented by those rows of the truth table corresponding to the rows for which the statement is false. For example, for Table 1.3, this corresponds to rows 1, 2 and 4 giving rise to the statement

$$A \cdot B \cdot C \vee A \cdot B \cdot \overline{C} \vee A \cdot \overline{B} \cdot \overline{C} \tag{1.29}$$

Negating this statement gets us back to the original statement which we wish to represent. We can negate (1.29) using De Morgan's laws (1.8) and (1.9). The effect is to create the logical dual of (1.29) which is

$$(\overline{A} \vee \overline{B} \vee \overline{C}) \cdot (\overline{A} \vee \overline{B} \vee C) \cdot (\overline{A} \vee B \vee C) \tag{1.30}$$

It can be checked (Exercise 1.7.3) that this statement (in ECNF) also represents Table 1.2. The rule for converting a truth table representation to a statement in ECNF is therefore to take each row of the truth table for which the statement is false, create a disjunction of the literals in it by negating those literals corresponding to a T entry and not negating those entries corresponding to an F entry and then taking the conjunction of the resultant (disjunctive) clauses.

If a statement is in non-extended DNF or CNF (i.e. each clause does not necessarily contain all literals) it is cumbersome to expand it into EDNF or ECNF in order to use the truth table representation to create the other normal form. Instead we can use De Morgan's laws to convert to the other form. We demonstrate this by the following example.

Example 1.3 Convert the following statement in DNF into CNF:

$$A \cdot \overline{B} \cdot C \ \vee B \cdot \overline{D} \ \vee A \cdot D \tag{1.31}$$

Using the distributive law (1.7) we combine one literal from each conjunctive clause into a disjunction in all possible ways and form the conjunction of these disjunctions (also using the simplifications (1.14) and (1.21)). This gives

$$(A \vee B) \cdot (A \vee B \vee D) \cdot A \cdot (A \vee \overline{B} \vee \overline{D}) \cdot (A \vee B \vee C) \cdot (A \vee C \vee D) \tag{1.32}$$

This statement is in far from its simplest form but the topic of 'simplification' is postponed until Chapter 4.

It is often much more economical to express in either DNF or CNF depending on the statement and application. Obviously, in order to do the reverse transformation and convert a statement in CNF into DNF, we can use the (logically dual) form of De Morgan's laws (see Exercise 1.7.4).

In order to show the potential complexity of such transformations we consider another example.

Example 1.4 Convert the following statement in DNF into CNF:

$$X_1 \cdot X_2 \vee X_3 \cdot X_4 \vee X_5 \cdot X_6 \vee \cdots \vee X_{2n-1} \cdot X_{2n} \tag{1.33}$$

This results in the following conjunction of disjunctions:

$$\begin{gathered}
(X_1 \vee X_3 \vee X_5 \ldots \vee X_{2n-3} \vee X_{2n-1}) \\
\cdot (X_1 \vee X_3 \vee X_5 \ldots \vee X_{2n-3} \vee X_{2n}) \\
\vdots \\
\cdot (X_2 \vee X_4 \vee X_6 \ldots \vee X_{2n-2} \vee X_{2n-1}) \\
\cdot (X_2 \vee X_4 \vee X_6 \ldots \vee X_{2n-2} \vee X_{2n})
\end{gathered} \tag{1.34}$$

It can be seen that instead of the n (conjunctive) clauses of 2 literals each we now have 2^n disjunctive clauses of n literals each. What's more no simplification is possible. Clearly, in this case, DNF is more compact. Also the amount of computation necessary to carry out the transformation is an exponential function of the number of variables. However, in other cases (e.g. the dual of (1.34)) CNF may be more compact. This is obviously an 'extreme' example but in more general cases the two forms may be very unbalanced in the size of their representations.

There is another, ingenious, approach to transforming statements in DNF to CNF and vice versa which reduces the computational complexity by introducing new literals. We demonstrate this by again using Example 1.4.

We introduce new literals $X_{i,j}$ representing the conjunctions $X_i \cdot X_j$ in (1.33). Equation (1.33) can now be written as

$$X_{1,2} \vee X_{3,4} \vee X_{5,6} \vee \cdots \vee X_{2n-3,2n-2} \vee X_{2n-1,2n} \tag{1.35}$$

i.e. as a disjunctive clause.

We must also model the conditions

$$X_{i,j} \longrightarrow X_i \cdot X_j \tag{1.36}$$

These may be written as

$$(\overline{X}_{i,j} \vee X_i) \cdot (\overline{X}_{i,j} \vee X_j) \tag{1.37}$$

Hence we have the conjunction of (1.35) with all the pairs of disjunctive clauses in (1.37) for each pair of i and j in each literal in (1.35). This is clearly a statement in CNF. It contains $3n$ literals and $2n + 1$ disjunctive clauses. One of them has n literals and the rest 2 literals each.

Exercise 1.7.6 involves converting the dual expression to (1.33), i.e. a statement in CNF, to a statement in DNF in an analogous manner.

1.3.4 Complete Sets of Connectives

We have shown that, by means of EDNF or ECNF, we can express any statement using only the connectives '$-$', '$\vee$' and '$\cdot$'. They are said to be a *complete* set of connectives in the sense that any compound statement can be written using them alone. It is, in fact, possible to suffice with only the set of connectives $\{-, \vee\}$.

In order to show this we can apply *De Morgan's* law (1.8) in order to replace all occurrences of '$\cdot$' in a statement by $-$ and $\vee$.

Similarly the set of connectives $\{-, \cdot\}$ form a complete set as can be seen by eliminating all occurrences of '$\vee$' in a statement using De Morgan's law (1.9).

However, it is also possible to represent all statements using a single connective. There are two such *complete connectives* (of two atomic statements). They are represented by the symbols '$\downarrow$' and '$|$' and are referred to as the *connective arrow* and the *Sheffer stroke*, respectively. They are defined in the following truth table (Table 1.4).

In intuitive terms '$\downarrow$' and '$|$' represent the logical connectives 'nor' and 'nand' ('not and'), respectively. They are sometimes manufactured as logical 'gates' for electrical circuits, as discussed in Chapter 4, in view of their completeness.

Table 1.4 Truth Tables for the Connective Arrow and Sheffer Stroke

A	B	$A \downarrow B$	$A \mid B$
T	T	F	F
T	F	F	T
F	T	F	T
F	F	T	T

In order to demonstrate their completeness we can show, by means of truth tables, that '$-$' , '$\cdot$' and '$\vee$' can be expressed using them. We can demonstrate (Exercise 1.7.7) the following equivalences:

$$\overline{A} \equiv A \downarrow A \tag{1.38}$$

$$A \cdot B \equiv (A \downarrow A) \downarrow (B \downarrow B) \tag{1.39}$$

$$A \vee B \equiv (A \downarrow B) \downarrow (A \downarrow B) \tag{1.40}$$

$$\overline{A} \equiv A|A \tag{1.41}$$

$$A \cdot B \equiv (A|B)|(A|B) \tag{1.42}$$

$$A \vee B \equiv (A|A)|(B|B) \tag{1.43}$$

Exercise 1.7.8 is to prove that these are the only possible complete connectives combining two variables. There are, however, many more complete connectives for combining three, four and more variables, but this is beyond the scope of this book.

1.3.5 The Calculus of Indications

This is an ingenious method of constructing the propositional calculus using one symbol only. The method has found some application in the design of electrical circuits. It also leads to great notational economy. The single symbol which we use is '$\rceil$'. The originator of the method intended it to stand for a '*distinction*' which he felt was the most fundamental useful concept possible. From this primitive notion he constructed his calculus. We will not discuss the philosophical aims of the originator but we will describe the mechanics of the system.

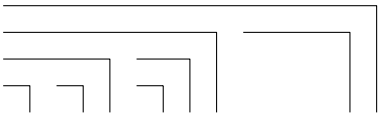
The symbol '$\rceil$' can be combined with itself in two ways:

$$\rceil\rceil \equiv \rceil \tag{1.44}$$

$$\rceil^{\rceil} \equiv \tag{1.45}$$

The 'empty' symbol in (1.45) is deliberate. Writing the symbol above itself in (1.45) cancels it out, whereas juxtaposing it in (1.44) results in itself.

If we have a structure such as

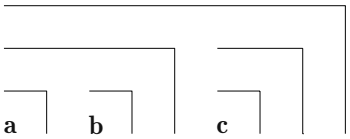


(1.46)

we can reduce it down to '⌉' or ' ' (the empty symbol). Any expression can be simplified down to one of these two symbols. Exercise 1.7.9 involves simplifying this and other expressions.

We can represent general expressions built up from '⌉' by (meta symbols) a,b, ... If known these expressions will simplify to '⌉' or ' '. We can give them the *interpretations* F and T, respectively, i.e. a,b, . . . can be interpreted as statements which are false or true according to whether they reduce to '⌉' or ' '.

The expressions a,b, . . . can be incorporated into enhanced expressions, e.g.



(1.47)

Here the symbol '⌉' can be interpreted as a connective applying to any number of statements. For example a ⌉ is a connective applying to one statement.

(1.48)

is a connective applying to two statements.

(1.49)

is a connective applying to three statements.

Since the expressions a,b,c,. . . each reduce to '⌉' (F) or ' ' (T) we can create truth tables for the application of '⌉'.

It can be verified that

a	a⌉
T	F
F	T

since if a reduces to (empty) (T) then a⌉ reduces to ⌉ (F). However, if a reduces to ⌉ (F) then a⌉ reduces to (T). Clearly, applied to one statement, ⌉ performs to role of the − connective.

Applying ⌝ to two statements it can be verified that

a	b	$\overline{\mathbf{a} \mid \mathbf{b}}$
T	T	F
T	F	T
F	T	T
F	F	T

This is the '|' (Sheffer stroke) (nand) connective defined above. As demonstrated there, it is a complete connective. It can be verified (Exercise 1.7.10) that '⌝' applied to any number of statements yields a complete connective for that number of statements. Identities (1.41) – (1.43) show how any statement can therefore be modelled using '⌝'. For example instead of modelling '·' in the form of (1.42) we can write it as ab .'∨' can be modelled $\overline{\mathbf{a} \mid \mathbf{b}}$ as in contrast to (1.43).

Exercise 1.7.11 involves using this 'multivalued' connective to represent a number of statements. It has the advantage that it is not always necessary to repeat the same literal (which also does not need to be negated) , as is necessary in expressions such as (1.42) and (1.43). One can also develop normal forms for compound statements using this symbol.

1.3.6 Venn Diagrams

These are a useful way of interpreting and representing compound statements. Each atomic statement corresponds to a region of the plane. We represent all statements by a 'universal' region drawn as the rectangle in Fig. 1.1.

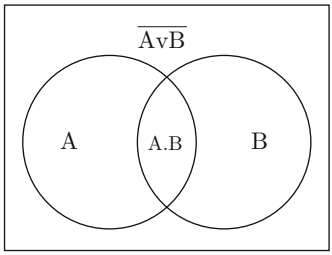


Fig. 1.1 A Venn diagram

The left-hand circle represents the statement A being true. The 'complementary' region to this circle represents $\overline{A}$ being true, i.e. A being false. The right-hand circle represents the statement B being true. The intersection of these circles represents $A \cdot B$ being true and the union of the circles represents $A \vee B$ being true, i.e. '∨' models the *union* (∪) operation of *Elementary Set Theory* and '·' models the *intersection* (∩) operation, while – models the *complement* operation.

Using Fig. 1.1 it is easy to verify, for example, De Morgan's laws. The region outside the two circles is modelled by $\overline{A \vee B}$. It is easy to see that this is also the intersection of the complement of A with the complement of B which is modelled by $\overline{A} \cdot \overline{B}$.

Figure 1.2 demonstrates how extended disjunctive normal form can model a statement.

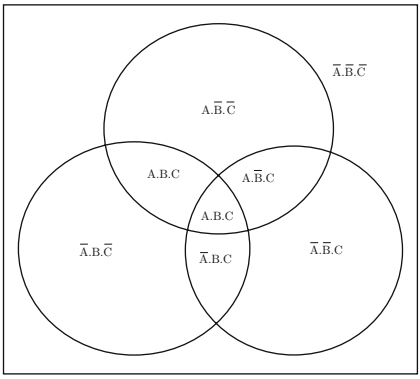


Fig. 1.2 A second Venn diagram

All eight disjoint regions are modelled by the eight conjunctive clauses made up from A, B, C and their negations. A is represented by the top circle, B by the left circle and C by the right circle. We have used the symbols A, B, C in two senses. On the figure they represent regions with the set operations '$\cup$', '$\cap$' and '$-$' (complement) applied. In the propositional calculus statements they represent atomic statements (indicating membership of the corresponding region) with the corresponding connectives '$\vee$', '$\cdot$' and '$-$' ('not') (using the same symbol as for the set complement).

1.4 The Predicate Calculus

While the propositional calculus is sufficient for many purposes it is not sufficient to formalise all mathematics. The predicate calculus was created for this purpose. In this system statements can be made about *objects* by means of *predicates*, e.g. $P(x, y)$. $P(x, y)$ could stand for a statement such as

$$`x \; is \; the \; father \; of \; y' \tag{1.50}$$

If the variables x and y are set to *constants* such as *'James'* and *'Mary'* we have the statement $P(James, Mary)$, which will be true or false meaning

$$`James \; is \; the \; father \; of \; Mary' \tag{1.51}$$

Predicates can be of one, two, three or any number of variables. The process of setting variables to constants is known as *instantiation.* Each variable in a predicate has a *domain* of possible values. In addition to the predicates we have the connectives of the propositional calculus for combining them into compound predicates. Instantiation of all the variables in the predicates of an expression converts it to a statement in the propositional calculus.

For example if we have the statement

$$(P(x, y) \vee Q(z)) \longrightarrow R(x, y, z) \tag{1.52}$$

we might instantiate it to

$$(P(a, b) \vee Q(c)) \longrightarrow R(a, b, c) \tag{1.53}$$

so long as a, b and c were in the appropriate domains. Its truth, or otherwise, will depend on the truth values of the atomic statements in the standard way.

1.4.1 The Use of Quantifiers

A major strength of the enhanced expressive power of the predicate calculus is the use of *quantifiers.* These are written as '$\forall$' and '$\exists$'. '$\forall$' means 'for all' and '$\exists$' means 'there exists'. If, for example, we have $\forall x\, Q(x)$ this means 'for all objects in the domain of x, $Q(x)$ is true'. This can then be regarded as an atomic proposition of the propositional calculus. Likewise the statement $\exists x\, Q(x)$ means 'there exists an object in the domain of x for which $Q(x)$ is true'.

The following statements are usually stipulated in the form of axioms of the predicate calculus but we simply state them as valid deductions and equivalences in our informal treatment. They obviously accord with the meanings we have given to the quantifiers. They enable us to manipulate and translate statements in the calculus (into, for example, normal forms). As before we assume that the objects (constants) $a, b, c, \ldots$ fall within the permitted domains of the variables $x, y, z, \ldots$ where they are substituted

$$\forall x\, Q(x) \Longrightarrow Q(a) \tag{1.54}$$

$$Q(a) \Longrightarrow \exists x\, Q(x) \tag{1.55}$$

$$\tilde{}\forall x\, Q(x) \equiv \exists x \tilde{} Q(x) \tag{1.56}$$

$$\tilde{}\exists x\, Q(x) \equiv \forall x \tilde{} Q(x) \tag{1.57}$$

Note that we are using the symbol '~' as an alternative to '$-$' solely for notational convenience. Equations (1.56) and (1.57) are obviously generalisations of De Morgan's laws.

If the domain of a predicate is finite (e.g. $a, b, \ldots, n$) then we can use the connectives '$\cdot$' and '$\vee$' in place of '$\forall$' and '$\exists$', e.g.

$$\forall x\, Q(x) \equiv Q(a) \cdot Q(b) \cdot \cdots \cdot Q(n) \tag{1.58}$$

$$\exists x\, Q(x) \equiv Q(a) \vee Q(b) \vee \cdots \vee Q(n) \tag{1.59}$$

Besides the ability to quantify over infinite domains (e.g. the natural, real or rational numbers) the new notation also has great advantages in helping us to create clear and succinct expressions even if the domains are finite. It can be very useful in *modelling* as is illustrated in Chapter 3.

The predicate calculus is sometimes referred to as 'first-order theory' since it allows us to quantify the objects within predicates but not 'higher order' objects such as the predicates themselves.

Each quantifier has a *scope* over which it applies, e.g.

$$\exists x(T(x, y) \vee U(x, z)) \cdot \tilde{} W(s, z) \tag{1.60}$$

is itself a predicate of the variables y, z, s. The scope of $\exists x$ is the first disjunction of T and U and is indicated by bracketing them. It does not extend to W. The variable x is said to be *bound* by the quantifier. It is a *dummy* variable which could equally well be replaced by another variable and the meaning be unchanged. To save confusion it is clearer not to use such a variable elsewhere, beyond the scope of the quantifier. Variables that are not bound are said to be *free.* y, z, s are free in the above expression. Since W does not involve x there would be no ambiguity in extending the scope of $\exists x$ to the whole expression by moving the last bracket after U to the end of the expression.

1.4.2 Prenex Normal Form

Using (1.56) and (1.57) we can move all the quantifiers to the beginning of an expression creating what is known as *prenex normal form*. We illustrate this by an example. The expression which is quantified will be known as the *core.*

Example 1.5 Transform the following expression into prenex normal form and the core into DNF:

$$\forall x_1(\exists x_2 R(x_1, x_2)) \longrightarrow \exists x_3(S(x_3, x_4, x_5) \cdot \forall x_6 T(x_4, x_6)) \tag{1.61}$$

Note that only variables x_1,x_2,x_3 and x_6 are bound. Therefore the expression is itself a predicate of the free variables x_4 and x_5. We can successively transform (1.61) into

$$\forall x_1(\exists x_2(R(x_1, x_2)) \longrightarrow \exists x_3(S(x_3, x_4, x_5) \cdot \forall x_6 T(x_4, x_6))) \tag{1.62}$$

$$\forall x_1(\exists x_2(R(x_1, x_2)) \longrightarrow \exists x_3 \forall x_6(S(x_3, x_4, x_5) \cdot T(x_4, x_6))) \tag{1.63}$$

$$\forall x_1 \exists x_2 \exists x_3 \forall x_6(R(x_1, x_2)) \longrightarrow (S(x_3, x_4, x_5) \cdot T(x_4, x_6)) \tag{1.64}$$

$$\forall x_1 \exists x_2 \exists x_3 \forall x_6(\tilde{} R(x_1, x_2)) \vee (S(x_3, x_4, x_5) \cdot T(x_4, x_6)) \tag{1.65}$$

It is sometimes convenient to allow one quantifier to apply to a group of variables, e.g. $\exists x_1 \exists x_2$ can be written as $\exists x_1 x_2$. Similarly for $\forall$. Also it can easily be verified that $\exists x_1 x_2$ means the same as $\exists x_2 x_1$. Similarly for $\forall$.

When reading a statement such as (1.65) it must be remembered that the quantifiers should be read from left to right. The left most quantifier applies to the statement (including quantifiers) to its right and so on. However, the order of $\exists$ and $\forall$ cannot be reversed. $\exists x \forall y P(x, y)$ does not mean the same as $\forall y \exists x P(x, y)$. To illustrate this let us take the domain of x and y as the integers and $P(x, y)$ as the *relation* $x \leqslant y$. The first statement is clearly not true in this interpretation since there is no integer which is smaller or equal to **all** the rest. However, the second statement is true in this interpretation since for any integer there is always an integer which is smaller or equal to it (e.g. itself). One might conveniently represent such an integer as x_y indicating that it depends on the value of y, whereas for the first statement we were seeking an x which was independent of the value of y.

In all interpretations of the predicate $P(x, y)$ and the domains of x and y, $\exists x \forall y P(x, y)$ is stronger than $\forall y \exists x P(x, y)$. We prove this below.

Example 1.6 Show that, in general,

$$\exists x \forall y P(x, y) \implies \forall y \exists x P(x, y) \tag{1.66}$$

If the premise holds then for all y there is an x such that $P(x, y)$, i.e. a particular, single, x serves the purpose for all the y. (For the positive integers, with the predicate '$x \leqslant y$' for example, 1 serves the purpose.) Hence for a particular y the conclusion will also be true since the specific x chosen for the premise will also serve the purpose.

In order to demonstrate the expressive power of the predicate calculus in a non-mathematical setting we consider the following example.

Example 1.7 Express the following statement using the predicate calculus:

You can fool all of the people some of the time
and some of the people all of the time, but
you cannot fool all of the people all of the time (1.67)

Let the predicate $P(x, y)$ be given the interpretation 'x can be fooled at time y'. The statement can then be written as

$$\exists v \forall u P(u, v). \forall v \exists u P(u, v). \tilde{} \forall v \forall u P(u, v) \tag{1.68}$$

Note the temptation to use the '$\longrightarrow$' symbol before the last quantified predicate. But this would not capture the true meaning.

1.5 Decidable Fragments of Mathematics

When the predicate calculus is used to formalise parts of mathematics it is necessary to append *functions* (e.g. '+' , 'log') and *constants* (e.g. $2, -1, \pi$) together with certain rules involving them (sometimes given in the form of extra axioms). *Relations* (e.g. '=', ' >', '$coprime$') are represented by predicates.

Although full arithmetic, formalised using the predicate calculus, is not decidable there are smaller 'theories' within it which are decidable. We illustrate this by two theories which are applicable to linear and integer programming.

1.5.1 The Theory of Dense Linear Order

The *theory of dense linear order* models order relations between variables and constants in a continuum such as the real or rational numbers. The '+' operation, the real numbers, the '=' and '<' relations (predicates) together with the transitivity rule $((x < y) \cdot (y < z) \Longrightarrow (x < z))$ are appended to the machinery of the predicate calculus. It is applicable to linear programming (LP). The constraints of an LP can be regarded as predicates with the LP variables as variables in these predicates.

It is easy to verify that we can express the '$\leqslant$' relation $((x < y) \vee (x = y))$. (Also we can express '$>$' and '$\geqslant$' but we choose to suffice with '$<$' and '$\leqslant$' for simplicity of exposition.) Repeated addition allows us to multiply variables by integers, e.g. $3x$ is given by $x + x + x$. Besides writing expressions such as $2x + 3y \leqslant 5$ we can include subtraction by, for example, writing $x - y \leqslant 3$ as $x \leqslant 3 + y$. Also we can allow multiplication by rational numbers by, for example, writing $\frac{2}{3}x < 5$ as $2x < 15$. For economy of representation we will use these derived relations and operations directly.

We are now in a position to make statements in this system and ask if they are true. This is a decidability problem and is illustrated by the following example.

Example 1.8 Is the following statement true ?

$$\begin{aligned} &\exists x \forall y (((x + y \leqslant 1) \vee (2x - 3y < 2) \vee (-x - 2y < -2)) \cdot \\ &\quad ((-x < -1) \vee (-2x - y < 2))) \\ &\quad x, y \epsilon \mathbb{R} \end{aligned} \tag{1.69}$$

We can use (1.56) and (1.57) to transform (1.69) to

$$\exists x \tilde{} \exists y ((-x - y < -1 \cdot -2x + 3y \leqslant -2 \cdot x + 2y \leqslant 2) \vee (x \leqslant 1 \cdot 2x + y \leqslant -2)) \tag{1.70}$$

Expressing this in the form

$$\begin{aligned}\exists x^{\sim}\ & [\exists y(-x - y < -1 \cdot -2x + 3y \leqslant -2 \cdot x + 2y \leqslant 2) \\ & \vee \exists y(x \leqslant 1 \cdot 2x + y \leqslant -2)]\end{aligned} \tag{1.71}$$

y can be eliminated from each of the two conjunctive clauses preceded by $\exists y$ by a process known as the *elimination of existential quantifiers.* The first clause can be written in the form

$$\exists y(-x + 1 < y \ \cdot \ y \leqslant \frac{2}{3}x - \frac{2}{3} \cdot y \leqslant -\frac{1}{2}x + 1) \tag{1.72}$$

We now consider the meaning of the above statement. It is that there exists a number from a continuum (e.g. the reals or the rationals) that lies strictly between $-x + 1$ and both of $\frac{2}{3}x - \frac{2}{3}$ and $-\frac{1}{2}x + 1$. The condition for this is that

$$-x + 1 < \frac{2}{3}x - \frac{2}{3} \tag{1.73}$$

and

$$-x + 1 < -\frac{1}{2}x + 1 \tag{1.74}$$

Clearly (1.72) implies (1.73) and (1.74). But also (1.73) and (1.74) together imply (1.72) since we could take y as any number greater than $-x+1$ and less than or equal to the minimum of $\frac{2}{3}x - \frac{2}{3}$ and $-\frac{1}{2}x + 1$. In general we have to consider all pairs of inequalities where the variable to be eliminated is on opposite sides of the inequality sign '$<$' (or '$\leqslant$'). If the quantified variable to be eliminated occurs in an equality relation the elimination is even easier. We can use the equality to substitute it out of all the other relations. Obviously we modify the resultant inequalities accordingly when the lower and upper inequalities before the elimination are different combinations of '$<$' and '$\leqslant$'. We rewrite (1.73) and (1.74) as

$$-\frac{5}{3}x < -\frac{5}{3} \tag{1.75}$$

and

$$-\frac{1}{2}x < 0 \tag{1.76}$$

The second conjunctive clause in (1.71) can be written in the form

$$\exists y(x \leqslant 1 \cdot y \leqslant -2x - 2) \tag{1.77}$$

Since there is no lower limit to y the second inequality is true for all x and may be ignored giving the equivalent statement

$$x \leqslant 1 \tag{1.78}$$

Statement (1.69) has therefore been reduced to

$$\exists x \tilde{} (-\frac{5}{3}x < -\frac{5}{3} \cdot -\frac{1}{2}x < 0 \vee x \leqslant 1) \tag{1.79}$$

which (using De Morgan's laws) can be written as

$$\exists x((\frac{5}{3}x \leqslant \frac{5}{3} \vee \frac{1}{2}x \leqslant 0) \cdot -x < -1) \tag{1.80}$$

or in CNF as

$$\exists x(\frac{5}{3}x \leqslant \frac{5}{3} \cdot -x < -1) \ \vee \ \exists x(\frac{1}{2}x \leqslant 0 \cdot -x < -1) \tag{1.81}$$

We now again eliminate the existential quantifier in both clauses to give

$$1 < 1 \vee 0 < -1 \tag{1.82}$$

Clearly both inequalities are false showing the original statement to be false.

1.5.2 Arithmetic Without Multiplication

We now consider a decision procedure for arithmetic without multiplication and present this in a form which is applicable to integer programming (IP) as discussed in Chapter 2.

As in the theory of dense linear order we consider all terms of the form nx as shorthand for $x + x + \cdots + x$ (n times). The theory looks very similar to the above theory except for the fact that the domain of the variables is now restricted to the integers. This makes the theory 'more difficult' in the sense that a decision procedure is more complicated. Again the constraints in an IP can be regarded as predicates. We illustrate the procedure by an example.

Example 1.9 Is the following statement true?

$$\exists xy(-2x - 2y \leqslant -7 \cdot 8x + 7y \leqslant 28 \cdot -x \leqslant -1) \tag{1.83}$$
$$x, y \epsilon \mathbb{Z}$$

i.e. where the domain of the variables is the integers.

In order to eliminate y and its existential quantifier we write the inner formula in the form

$$\exists y(7(-2x+7) \leqslant 14y \leqslant 2(-8x+28) \cdot -x \leqslant -1) \tag{1.84}$$

It is not now sufficient to observe that the expression to the left of the first inequality is less than or equal to that on the right. There must also be a multiple of 14 between them. This can be captured in a *finite* way by writing

$$\begin{aligned} 7(-2x+7)+s \leqslant 2(-8x+28) \cdot 7(-2x+7)+s \equiv 0 \,(\text{mod } 14) \\ \cdot -x \leqslant -1 \end{aligned} \tag{1.85}$$

where $s = 0, 1, \ldots, 13$

Since the left-hand expressions are multiples of 7 (by the congruence) , s must be also and can be written as $7t$ allowing (1.85) to be simplified to

$$7(-2x+7)+7t \leqslant 2(-8x+28) \cdot -2x+t \equiv 1 \,(\text{mod } 2) \tag{1.86}$$

where $t = 0, 1$

Clearly the congruence can only be satisfied if $t = 1$. Hence the inequality reduces to

$$2x \leqslant 0 \tag{1.87}$$

Equation (1.83) has therefore been reduced to

$$\exists x(2x <= 0 \cdot -x \leqslant -1) \tag{1.88}$$

which can be written as

$$\exists x(2 \leqslant 2x \leqslant 0) \tag{1.89}$$

Strictly following our rules we continue the procedure by eliminating x and its quantifier to give

$$2+w <= 0 \cdot 2+w \equiv 0 \,(\text{mod } 2) \tag{1.90}$$

where $w = 0, 1$.

Clearly the congruence implies $w = 0$ showing (1.90) to be false. Hence (1.83) is false.

Exercise 1.7.16 is to show that (1.83) is true when the variables are 'relaxed' to be real (or rational).

It should be pointed out that this is a particularly simple example. In general we cannot eliminate the implied congruences at each stage. They have to

be incorporated in the subsequent eliminations. Full details of the method, when applied to integer programming models, are referenced in Sect. 2.6.

1.6 References and Further Work

A good introduction to logic is Mendelson [81]. Another text is Shoenfield [101] . Also Langer [72] is a very clear text. Russell and Whitehead [96] give the results of their formalisation. Gödel [44] presents his major results. A 'popular' description of Gödel's work is Nagel and Newman [85].

The propositional calculus (Boolean Algebra) is due to Boole [18]. Truth tables were invented by Wittgenstein [124] and Post [90] who also investigated complete connectives. The Sheffer stroke is due to Sheffer [99] and the connective arrow due to Peirce [89].

The compact way of converting statements from DNF to CNF is due to Tseitin [107]. Wilson [123] also presents the method.

The calculus of indications is due to Spencer-Brown [103].

The predicate calculus is usually attributed to Frege [39].

The decision procedure for the theory of dense linear order is due to Langford [73] and that for arithmetic without multiplication is due to Presburger [91].

1.7 Exercises

1.7.1 Use a truth table to show that the following statement is a tautology:

$$((A \vee B) \cdot (A \longrightarrow C) \cdot (B \longrightarrow C)) \longrightarrow C$$

1.7.2 Verify, by means of truth tables the equivalences (1.4) – (1.21).

1.7.3 Check that (1.30) represents the statement in Table 1.2.

1.7.4 Use De Morgan's laws to convert the following statement to DNF:

$$(\overline{A} \vee B \vee \overline{C}) \cdot (B \vee C \vee \overline{D}) \cdot (A \vee \overline{B} \vee \overline{D})$$

1.7.5 Convert the statement in 1.7.4 into DNF by means of the distributive laws and verify, by means of truth tables, that the statements produced in 1.7.4 and 1.7.5 are equivalent.

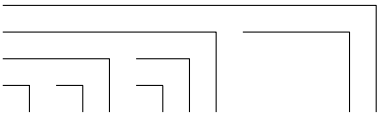
1.7.6 Convert the following statement into DNF by introducing new variables to represent the disjunctive pairs of statements:

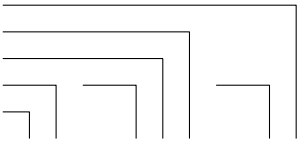
$$(X_1 \vee X_2) \cdot (X_3 \vee X_4) \cdot (X_5 \vee X_6) \cdots (X_{2n-1} \vee X_{2n})$$

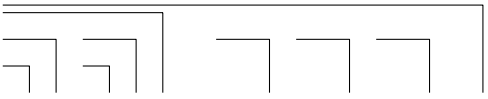
1.7.7 Verify, by means of truth tables, (1.38) – (1.43).

1.7.8 Show that '|' and '$\downarrow$' are the only possible complete connectives of two variables.

1.7.9 Simplify the following expressions:

i. 

ii. 

iii. 

1.7.10 Show that $\neg$ applied to any number of statements is a complete connective.

1.7.11 Represent the following statements using only the connective $\neg$:

i. $(A \vee \overline{B} \vee C) \rightarrow (D \vee \overline{E})$

ii. $(A \longrightarrow (\overline{B} \vee C)) \leftrightarrow (B \cdot A)$

iii. $(A \cdot B \cdot \overline{C}) \vee B \cdot A)$

1.7.12 Represent the statement 1.7.11 (i) using a Venn diagram.

1.7.13 Represent the following statement using the predicate calculus:

Some animals are mammals but not all animals are mammals although all mammals are animals.

1.7.14 Represent the following statement by the predicate calculus:

Every even number greater than 2 *is expressible as the sum of two prime numbers.*

Even numbers and prime numbers should be expressed in terms of more elementary predicates.

1.7.15 Is the following statement true?

$$\exists x y \forall z (x + y <= z \cdot 2x - 3y \leqslant z \cdot x > z)$$

if (i) $x, y \epsilon \mathbb{R}$ (ii) $x, y \epsilon \mathbb{Z}$.

1.7.16 Show that (1.83) of Example 1.9 is true if the variables are real (or rational).

Chapter 2
Integer Programming

In this chapter we begin with a brief explanation of linear programming (LP) since integer programming (IP) is usually regarded as an extension of LP. Also most practical methods of solving IP models rely on solving an LP model first. However, our discussion of LP will be brief since this is not the main theme of this book. Although LP models are often solved in the course of solving IP models we will, in the main, regard a method of solving LP models in this context as a 'black box'. However, we will give a (computationally inefficient) method of solving LP models which illustrates the use of the predicate calculus as discussed in Sect. 1.5 as well as demonstrating important properties of LP models which are relevant to IP. In Sect. 2.4 we discuss the comparative computational complexity of IP compared with LP. A list of references to the further study of LP is given in Sect. 2.5.

2.1 Linear Programming

A linear programme (LP) is a problem of maximising or minimising a *linear* expression subject to a number of *linear* constraints which take the form of linear expressions being less than or equal to '$\leqslant$', greater than or equal to '$\geqslant$' or equal to '$=$' given numbers. For example the following is an LP model.

Example 2.1 A linear programme

Maximise

$$2x_1 + 3x_2 - x_3 \tag{2.1}$$

subject to

$$x_1 + x_2 \leqslant 3 \tag{2.2}$$

$$-x_1 + 2x_3 \geqslant -2 \tag{2.3}$$

$$-2x_1 + x_2 - x_3 = 0 \tag{2.4}$$

H.P. Williams, *Logic and Integer Programming,* International Series in Operations Research & Management Science 130, DOI 10.1007/978-0-387-92280-5_2,
© Springer Science+Business Media, LLC 2009

$$x_1, x_2, x_3 \geqslant 0 \tag{2.5}$$

$$x_1, x_2, x_3 \epsilon\ \mathbb{R}$$

It is usual, but not necessary, for the variables to be restricted to be non-negative by the last set of inequalities. Sometimes LPs are expressed in *standard* form as maximisations subject to all '$\leqslant$' constraints (apart from the non-negativity constraints) or alternatively as minimisations subject to all '$\geqslant$' constraints. The expression to be maximised or minimised is known as the *objective function*. Clearly it is possible to convert a maximisation to a minimisation (and vice versa) by negating the objective function. '$\leqslant$' constraints can be converted to '$\geqslant$' constraints by negating throughout (and vice versa). '$=$' constraints can be converted to a pair of a '$\leqslant$' and a '$\geqslant$' constraint simultaneously. We convert Example 2.1 to a standard form by the following example.

Example 2.2 Convert Example 2.1 to a standard form

This becomes

Maximise

$$2x_1 + 3x_2 - x_3 \tag{2.6}$$

subject to

$$x_1 + x_2 \leqslant 3 \tag{2.7}$$

$$x_1 - 2x_3 \leqslant 2 \tag{2.8}$$

$$-2x_1 + x_2 - x_3 \leqslant 0 \tag{2.9}$$

$$2x_1 - x_2 + x_3 \leqslant 0 \tag{2.10}$$

$$x_1, x_2, x_3 \geqslant 0 \tag{2.11}$$

$$x_1, x_2, x_3 \epsilon\ \mathbb{R}$$

After converting the '$=$' to a '$\leqslant$' and a '$\geqslant$', the '$\geqslant$' has been negated throughout to make it a '$\leqslant$'.

While a standard form is useful, for explanatory purposes, in practice models are usually kept in a more general form such as in Example 2.1.

In order to make an LP model more precise we express Example 2.1 using the predicate calculus.

Example 2.3 Express Example 2.1 using the predicate calculus and solve it by the elimination of existential quantifiers

This becomes

Find the maximum value of z such that

$$\begin{aligned}\exists z x_1 x_2 x_3\ (z - 2x_1 - 3x_2 + x_3 &= 0\\ \cdot\, x_1 + x_2 &\leqslant 3\\ \cdot x_1 - 2x_3 &\leqslant 2\\ \cdot - 2x_1 + x_2 - x_3 &= 0\\ \cdot - x_1 &\leqslant 0\\ \cdot - x_2 &\leqslant 0\\ \cdot - x_3 &\leqslant 0)\end{aligned} \tag{2.12}$$

$$x_1, x_2, x_3 \epsilon\ \mathbb{R}$$

By means of the first equation z has been set equal to the objective function (2.6).

We have deliberately quantified the variable z as we wish to test if the model has a *feasible* solution before seeking the maximum z, if it exists, i.e. we are first of all finding if the statement is *true*.

Eliminating x_1 using the first equation, by the method described in Example 1.8, we obtain

$$\begin{aligned}\exists z x_2 x_3 (z - x_2 + x_3 &\leqslant 6\\ \cdot z - 3x_2 - 3x_3 &\leqslant 4\\ \cdot - z + 4x_2 - 2x_3 &= 0\\ \cdot - z + 3x_2 - x_3 &\leqslant 0\\ \cdot - x_2 &\leqslant 0\\ \cdot - x_3 &\leqslant 0)\end{aligned} \tag{2.13}$$

Eliminating x_2 using the remaining equation

$$\begin{aligned}\exists z x_3 (3z + 2x_3 &\leqslant 24\\ \cdot z - 18x_3 &\leqslant 16\\ \cdot - z + 2x_3 &\leqslant 0\\ \cdot - z - 2x_3 &\leqslant 0\\ \cdot - x_3 &\leqslant 0)\end{aligned} \tag{2.14}$$

Eliminating x_3 between the inequalities (and carrying out some simplification, including removing obvious redundancies) we obtain

$$\begin{aligned}\exists z (z &\leqslant 8\\ \cdot - z &\leqslant 0)\end{aligned} \tag{2.15}$$

Eliminating z we obtain

$$0 \leqslant 8 \tag{2.16}$$

which is clearly true, demonstrating that the original statement (2.12) is true (i.e. the LP is feasible). The maximum value of z satisfying (2.15) is 8. Setting $z = 8$ the first inequality in (2.15) is satisfied as an equation. Therefore so must be the first and last inequalities in (2.14), from which it is derived, making $x_3 = 0$. These inequalities, in turn, are derived from the equation and first inequality in (2.13) making $x_2 = 2$. These are derived from the two equations in (2.12) making $x_1 = 1$.

Hence we have the optimal solution to the LP in Example 2.1

$$x_1 = 1, \quad x_2 = 2, \quad x_3 = 0, \quad \text{Objective} = 8 \tag{2.17}$$

The above method of solving LPs by successively eliminating existential quantifiers is not the most efficient algorithm. Other algorithms, in particular the *Simplex* algorithm, are mentioned in Sect. 2.5 and references given. It is, however, worth remarking that the above method shows LP to be a decidable theory, which was the purpose for which the decision procedure was designed. The simplex algorithm, as originally stated, does not do this as it cannot be guaranteed to terminate. It can, however, be modified to guarantee convergence. References are given in Sect. 2.5. The method described here is usually referred to as Fourier–Motzkin elimination. Although not computationally efficient it does prove to be a particularly clear method of demonstrating important properties of LPs. Further algorithmic discussion is beyond the scope of this book but is fully discussed in the many texts on LP. We do, however, discuss those aspects of LP which are relevant to our main purpose of modelling and solving integer programming (IP) models.

2.1.1 The Dual of an LP Model

In Example 2.3 we obtained the optimal objective value by successively adding or subtracting the equality constraints, in suitable multiples, from the other constraints to eliminate variables and adding the '$\leqslant$' inequality constraints in suitable non-negative multiples. The net effect is to express the objective function as a linear combination of the constraints in such a way as to eliminate the variables. It can be verified that this results in multipliers of $1, \frac{8}{3}, 0, \frac{1}{3}, 0, 0, \frac{2}{3}$, respectively, on the constraints which, when added together in these multiples, produces the constraint $z \leqslant 8$. The effect of the multipliers is to eliminate all the variables (except z) and produce an upper bound on the possible objective values. We can formalise this in terms of the original model in Example 2.1, as seeking multipliers $y_1, y_2, y_3, v_1, v_2, v_3$ on all the constraints (2.2)–(2.5) so as to

Minimise

$$3y_1 - 2y_2 \tag{2.18}$$

subject to

$$y_1 - y_2 - 2y_3 - v_1 = 2 \tag{2.19}$$

$$y_1 + y_3 - v_2 = 3 \tag{2.20}$$

$$2y_2 - y_3 - v_3 = -1 \tag{2.21}$$

$$y_1 \geqslant 0, \quad y_2 \leqslant 0, \quad v_1, v_2, v_3 \geqslant 0 \tag{2.22}$$

$$y_1, y_2, y_3, v_1, v_2, v_3 \in \mathbb{R}$$

Notice that y_2 is constrained to be non-positive and y_3 is not constrained in sign. This is because the second constraint is of the form '$\geqslant$' (and therefore effectively subtracted from the other constraints if not converted to the '$\leqslant$' form as is done in Examples 2.2 and 2.3) and the third constraint is an equation which may be added or subtracted from the other constraints. By minimising expression (2.18) we are seeking the *smallest* right-hand side b for expressions of the form $z \leqslant b$ which are implied by the original constraints. This smallest value of b is obviously the maximum z which can be obtained.

The variables v_i (the multipliers on the non-negativity constraints (2.5) stated in the form of '$\leqslant$' constraints) are known as *surplus* variables. We can ignore them if we rewrite (2.18)–(2.22) as follows:

Minimise

$$3y_1 - 2y_2 \tag{2.23}$$

subject to

$$y_1 - y_2 - 2y_3 \geqslant 2 \tag{2.24}$$

$$y_1 + y_3 \geqslant 3 \tag{2.25}$$

$$2y_2 - y_3 \geqslant -1 \tag{2.26}$$

$$y_1 \geqslant 0, \quad y_2 \leqslant 0 \tag{2.27}$$

$$y_1, y_2, y_3 \in \mathbb{R}$$

The model given by (2.23)–(2.27) is, of course, itself an LP model. It is known as the *dual* model. In contrast the original model, given in Example 2.1, is known as the *primal* model. If the primal model were in standard form as a maximisation subject to all the constraints being '$\leqslant$' (apart from the non-negativity constraints on

the variables) then the dual model would also be in standard form as a minimisation subject to all '$\geqslant$' constraints.

The decision procedure, which we illustrated in Example 2.3, shows that the optimal solution to the dual model gives an attainable upper bound on the optimal objective value of the primal model. Hence, if both the primal and dual models have finite optimal solutions, their optimal objective values are the same (there is no '*duality gap*'). The optimal values of the dual variables are known as the *dual values* of the corresponding constraints in the primal model. It might be the case that the dual model is infeasible in which case the primal model may have no bound on its optimal objective value (it is said to be *unbounded)* or it might itself be infeasible. If the primal model is unbounded then we can produce no upper bound on the primal objective value and therefore the dual model can have no solution and must be infeasible. All these results are part of the *duality theorem* of LP. The variables in the dual model have important economic interpretations in many applications as well as having computational applications. Of course duality is also of mathematical interest since there is a symmetry between the primal and dual models. The dual of the dual model is the primal model (Exercise 2.6.3).

We should point out that the duality between LP models, which we have exhibited here, is different from the logical duality which we discussed in Chapter 1.

2.1.2 A Geometrical Representation of a Linear Programme

The variables in an LP model can be interpreted as the coordinates of points in space. The dimension of the space is equal to the number of variables in the model. We illustrate this by giving a geometrical representation of the model in Example 2.1.

As all the variables are non-negative we restrict ourselves to the non-negative orthant. The constraints, in this example, restrict us to the triangular region A, B, C. This is known as the *feasible region.* Particular values for the objective function lie on planes with the orientation of that shown. Increasing values of the objective function arise as the planes are moved in the direction shown. However, we need a plane that intersects the feasible region. This happens, for this example, when an objective plane intersects the vertex B of the feasible region, whose coordinates $x_1 = 1, x_2 = 2, x_3 = 0$ therefore give the optimal solution (Fig. 2.1).

In this particular example the feasible region is *non-full dimensional*, i.e. not of a dimension equal to the number of variables. It illustrates a number of features of the geometrical representation of LPs which generalise.

The feasible region is a *polytope.* These are regions of space, in any number of dimensions, whose boundaries are *hyperplanes*, i.e. lines, planes or higher dimensional generalisations. They are of dimension one less than the dimension of the polytope. In this example the polytope of the feasible region is closed. In general, however, LP polytopes can be open, as illustrated by Example 2.5. Such polytopes are sometimes referred to as *polyhedra.* For an LP polytope the feasible region can be characterised by its boundaries, known as its *facets* or, alternatively by its *vertices.*

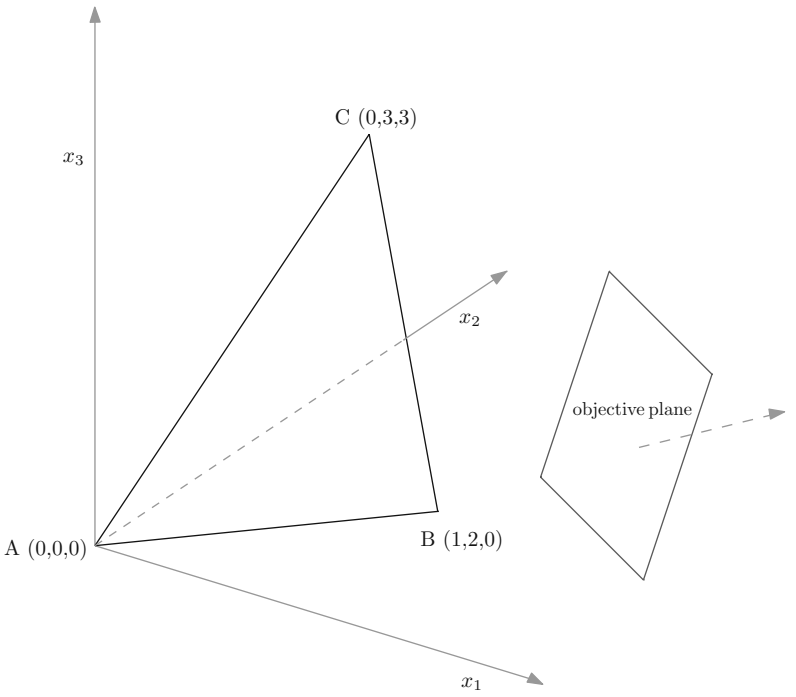


Fig. 2.1 A non-full dimensional LP model

Different values of the objective function will be represented by hyperplanes in the space. Moving to parallel hyperplanes in a direction orthogonal to the hyperplanes will increase the objective value and in the other direction decrease the objective value. As in the example the optimal solution will normally correspond to where an objective hyperplane intersects a vertex of the feasible region. However, it may be the case that the orientation of the objective hyperplane is such that there are *alternative optimal solutions* not at vertices. But among the alternative optimal solutions there will still be vertex solutions (so long as the model is not infeasible or unbounded).

Some of these different features of LP models are illustrated in examples below and in exercises in Sect. 2.6. We emphasise again that our purpose is to explain the nature of LP in so much as is necessary for the understanding of IP. There are many other texts that give a rigorous mathematical derivation of these properties, some of which are referenced in Sect. 2.5.

We give some other examples of LP models

Example 2.4 A full-dimensional LP model

Maximise

$$-4x_1 + 5x_2 + 3x_3 \tag{2.28}$$

subject to

$$-x_1 + x_2 - x_3 \leqslant 2 \tag{2.29}$$

$$x_1 + x_2 + 2x_3 \leqslant 3 \tag{2.30}$$

$$x_1, x_2, x_3 \geqslant 0 \tag{2.31}$$

$$x_1, x_2, x_3 \epsilon \mathbb{R}$$

This is illustrated in Fig. 2.2. Note that this model is in standard form.

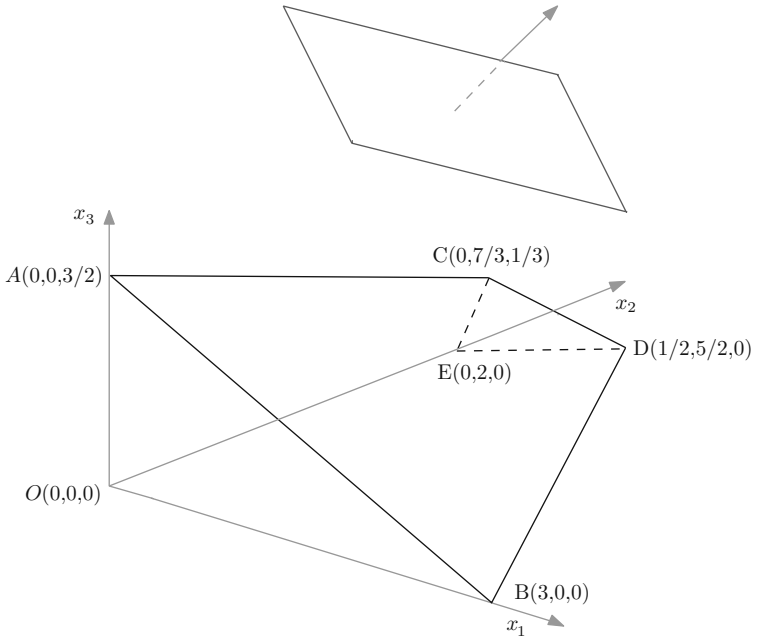


Fig. 2.2 A full-dimensional LP model

It can be seen that the feasible region has six vertices O, A, B, C, D and E with their coordinates marked. The orientation of the objective plane shows that the (unique) optimal solution arises from vertex C giving the solution $x_1 = 0$, $x_2 = \frac{7}{3}$, $x_3 = \frac{1}{3}$, objective $= \frac{38}{3}$.

Example 2.5 An LP with an open feasible region
Minimise

$$-x_1 + x_2 \tag{2.32}$$

subject to

$$x_1 + 2x_2 \geqslant 5 \tag{2.33}$$

$$-2x_1 + x_2 \geqslant -2 \tag{2.34}$$

$$x_1, x_2 \geqslant 0 \tag{2.35}$$

$$x_1, x_2 \epsilon \mathbb{R}$$

Note that this model is also in standard form. It is illustrated in Fig. 2.3. (When we have a minimisation model we adopt the convention that dual values on '$\geqslant$' constraints are non-negative, whereas dual values on '$\leqslant$' constraints are non-positive, in contrast to these conventions being reversed for maximisation models.)

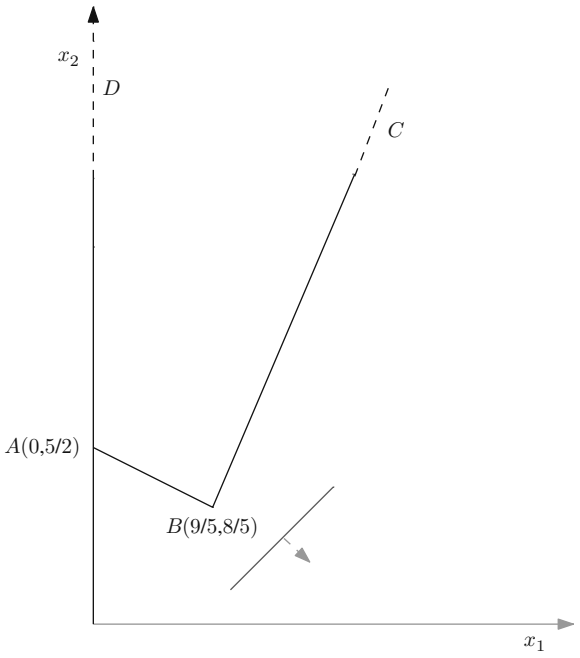


Fig. 2.3 An LP model with an open feasible region

The optimal solution occurs at vertex B giving $x_1 = \frac{9}{5}$, $x_2 = \frac{8}{5}$, objective $= -\frac{1}{5}$. If, however, the objective had been (say)

Minimise

$$-3x_1 + x_2 \tag{2.36}$$

then the model would have been unbounded since feasible solutions of ever-decreasing objective value could be found.

A polyhedron with an open feasible region (in any number of dimensions) is characterised by its vertices *and* its *extreme rays*. In the example AD and BC are extreme rays. If an LP is unbounded then the unboundedness is represented by an extreme ray such as BC, for this example, with objective (2.36). Note that points on this extreme ray give ever-decreasing values for objective (2.36).

It has already been remarked that if an LP is unbounded the corresponding dual model is infeasible, i.e. the decision procedure illustrated in Example 2.3 will produce no lower bound (in the case of a minimisation) on the objective (see Exercise 2.6.5). We give the dual model to Example 2.3 (with objective (2.36)).

Example 2.6 An infeasible LP model

Maximise

$$5y_1 - 2y_2 \tag{2.37}$$

subject to

$$y_1 - 2y_2 \leqslant -3 \tag{2.38}$$

$$2y_1 + y_2 \leqslant 1 \tag{2.39}$$

$$y_1, y_2 \geqslant 0 \tag{2.40}$$

$$y_1, y_2 \epsilon\, \mathbb{R}$$

In Fig. 2.4 it is shown that the constraints above are self-contradictory.

In order to satisfy constraint (2.38) (and the non-negativity constraints) we have to lie in the upper open region, but in order to satisfy constraint (2.39) we have to lie in the lower triangular region. These regions have no points in common.

Note that the property of infeasibility is independent of the objective function. If a model is infeasible it has no solutions irrespective of the objective.

It has already been remarked that if a model is unbounded its dual must be infeasible. Exercise 2.6.6 is to show that the dual of Example 2.6 (which is infeasible) is Example 2.5 with objective (2.36) (which is unbounded). It was also, however, remarked that if a model is infeasible there is another possibility: its dual may also be infeasible. Exercise 2.6.7 demonstrates this.

The method of solving LP models by the elimination of existential quantifiers, illustrated in Example 2.3, can be regarded in a geometrical context as a method of *projection.* Each time a variable is eliminated the model is projected down to an equivalent model in a space of one less dimension. The method can be valuable in reformulating both LP and IP models.

LP models arise in many contexts such as production, distribution, blending, oil refinery scheduling and chemical processing to name only a few. Models can involve millions of constraints and variables. In Sect. 2.5 references are given to comprehensive sources which discuss practical applications.

Before discussing IP we should point out that there are important classes of apparent IP models which can be solved as LPs as this leads to integer values for the

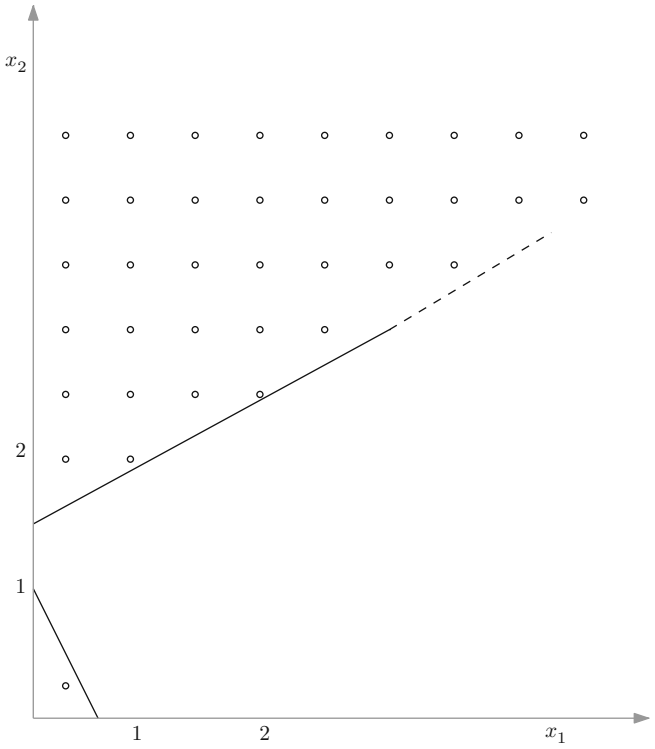


Fig. 2.4 Constraints defining an infeasible LP model

variables. In particular the *minimum cost network flow* model (with integer flows in and out of the network) and special cases such as the *transportation* and *assignment* problems have this property. They will not be discussed further here but references given in Sect. 2.5 discuss them.

2.2 Integer Programming

If all or some of the variables in an, otherwise, LP model are restricted to take *integer* values then we have a pure or a mixed IP (MIP) model, respectively. As in other branches of mathematics models involving integers are much more difficult to solve than models only involving real numbers. In Sect. 2.4 we discuss the issue of *computational complexity* and point out that IP models are generally much more difficult to solve than comparably sized LP models.

In Sect. 2.3 we point out that in most practical models the integer variables are restricted to the two values 0 and 1. Hence the connection with the propositional calculus. In this section we confine our attention to solving pure IP models with general integer variables. It will be seen that the methods effectively specialise to

the pure and mixed 0–1 case. For completeness we solve an illustrative model by the decision procedure illustrated in Example 1.9. Then we discuss the more efficient methods employed in practice. We consider the following example.

Example 2.7 Solve the following IP model using the decision procedure for arithmetic without multiplication

Maximise

$$x_1 + x_2 \tag{2.41}$$

subject to

$$2x_1 + 2x_2 \geqslant 3 \tag{2.42}$$

$$-2x_1 + 2x_2 \leqslant 3 \tag{2.43}$$

$$4x_1 + 2x_2 \leqslant 19 \tag{2.44}$$

$$x_1, x_2 \geqslant 0 \tag{2.45}$$

$$x_1, x_2 \epsilon\, \mathbb{Z}$$

We write the model in the form
Maximise z
such that

$$\begin{aligned} \exists z x_1 x_2 (z - x_1 - x_2 = 0 \\ \cdot\ -2x_1 - 2x_2 \leqslant -3 \\ \cdot\ -2x_1 + 2x_2 \leqslant 3 \\ \cdot\ 4x_1 + 2x_2 \leqslant 19 \\ \cdot\ -x_1 \leqslant 0 \\ \cdot\ -x_2 \leqslant 0) \end{aligned} \tag{2.46}$$

$$z, x_1, x_2 \epsilon\, \mathbb{Z}$$

Eliminating x_1 using the equation gives

$$\begin{aligned} \exists z x_2 (-2z \leqslant -3 \\ \cdot\ -2z + 4x_2 \leqslant 3 \\ \cdot\ 4z - 2x_2 \leqslant 19 \end{aligned} \tag{2.47}$$

$$\begin{aligned} &\cdot\ -z + x_2 \leqslant 0 \\ &\cdot\ -x_2 \leqslant 0) \end{aligned}$$

Since the variable eliminated (x_1) has a coefficient of ∓ 1 in the equation in (2.46) no congruences are needed in (2.47).

We rewrite this in the form

$$\begin{aligned} \exists z x_2(-2z &\leqslant -3 \\ \cdot\ 4x_2 &\leqslant 2z + 3 \\ \cdot\ 2(4z - 19) &\leqslant 4x_2 \\ \cdot\ 4x_2 &\leqslant 4z \\ \cdot\ 0 &\leqslant 4x_2) \end{aligned} \tag{2.48}$$

Eliminating x_2 reduces the system to

$$\begin{aligned} \exists z(-2z &\leqslant -3 \\ \cdot\ 2(4z - 19) + 2s &\leqslant 2z\ + 3 \\ \cdot\ 2(4z - 19) + 2s &\leqslant 4z \\ \cdot\ 0 &\leqslant 2z + 3 \\ \cdot\ 0 &\leqslant 4z \\ \cdot\ 4z - 19 + s &\equiv 0(\text{mod } 2)) \\ s &\in \{0, 1\} \end{aligned} \tag{2.49}$$

The congruence demonstrates that $s = 1$.

Therefore (2.49) (after simplification) can be rewritten as

$$\exists z(2z \leqslant 13 \cdot -2z \leqslant -3) \tag{2.50}$$

Eliminating z (and simplifying) gives

$$\begin{aligned} 3 + t &\leqslant 13 \cdot 3 + t \equiv 0(\text{mod } 2) \\ t &\in \{0, 1\} \end{aligned} \tag{2.51}$$

In order to satisfy the congruence $t = 1$. The inequality in (2.51) is then obviously true showing the IP to be feasible.

The maximum value of z satisfying (2.50) is 6 arising from the first inequality in (2.50). This inequality arises from the second and third inequalities in (2.48)

showing that $10 \leqslant 4x_2 \leqslant 15$ making $x_2 = 3$. These inequalities in turn arise from the equation and third and fourth of the inequalities in (2.47). The (unique) solution is therefore

$$x_1 = 3, \quad x_2 = 3, \quad \text{objective} = 6 \tag{2.52}$$

We now use the above example to illustrate the method usually used in practice for solving IP models. It is usually called the '*branch-and-bound*' algorithm for reasons which will become apparent.

2.2.1 The Branch-and-Bound Algorithm

Example 2.8 Solve the model in Example 2.7 by the branch-and-bound algorithm.

The first step in this method is to remove the integrality requirements and solve the resultant LP model. This is referred to as the *LP relaxation*. LP is computationally 'easy' compared with IP, as discussed in Sect. 2.5. If the optimal solution to the LP relaxation turns out to be integral then this will also be an optimal solution to the IP. There is an important class of models where this will always be the case. However, for this example, we obtain the fractional solution

$$x_1 = 2\frac{2}{3}, \quad x_2 = 4\frac{1}{6}, \quad \text{objective} = 6\frac{5}{6} \tag{2.53}$$

We now choose one of the integer variables (x) whose value has come out fractional (e.g. $N + f$ where N is an integer and $0 < f < 1$). There is no loss of generality in stipulating that *either* $x \leqslant N$ or $x \geqslant N + 1$ since x is not permitted to take any fractional value between N and $N + 1$. Both these possibilities rule out the current fractional solution. For the example we choose the *dichotomy* $x_1 \leqslant 2$ or $x_1 \geqslant 3$. The choice of variable x_1 over x_2 is arbitrary here. In practice it can be made *heuristically*, i.e. according to plausible (but not guaranteed) rules which might be hoped to produce the optimal integer solution as quickly as possible. For example it might be thought that since x_1 is further from its closer integer than x_2 this will have more effect than choosing x_2. It is convenient to represent this process as a *branching* in a tree as shown in Fig. 2.5.

At each node we give the solution of the corresponding LP relaxation. Node 2 corresponds to the original model together with the appended constraint $x_1 \leqslant 2$. Node 3 corresponds to the original model together with the appended constraint $x_1 \geqslant 3$. We number the nodes in order of the creation of the corresponding models. Besides the choice of *branching variable* there is also a choice of order in which to solve the LP relaxations of the nodes created. It is convenient to solve each of the two resultant LP models (the '*son*' and '*daughter*') immediately after their creation. It can be seen that the objective values corresponding to the LP relaxations of the two nodes have both got 'worse' (declined for a maximisation or increased for a

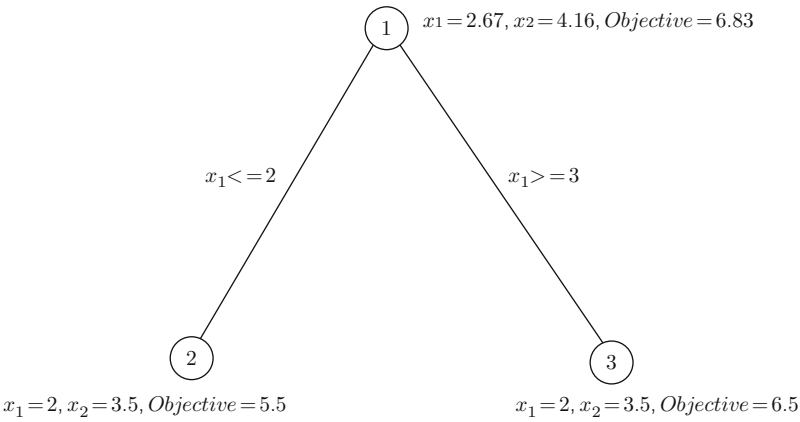


Fig. 2.5 A branch in the branch-and-bound algorithm

minimisation). This would be expected since extra constraints are being added. In fact they might not change if there were alternate solutions.

At this stage we have a choice of *waiting nodes* to 'develop', i.e. choose which node to branch from. This choice can again be made arbitrarily or heuristically. For example we might choose to develop node 3 in preference to node 1 since its objective has declined less (and there is therefore more chance of a better integer solution down this branch). For our example we continue this process and illustrate it in Fig. 2.6.

At node 5 the LP relaxation (and therefore the corresponding IP) is infeasible. This result is likely to happen as we are adding progressively stronger constraints down a branch. Once a node has become infeasible there is no sense in developing it further. It is said to have been *fathomed.* Node 4 is then developed in preference to the other waiting node 2 if we use the heuristic that the objective of the LP relaxation at node 4 is better than that at node 2. At node 6 we obtain an integer solution with an objective value of 6. Again there is no sense in developing this node further which is also said to have been fathomed. Since the objective value is better than that at node 2 no better integer solution can be obtained by developing that node. It is also, therefore, said to have been fathomed. The value of the best integer solution found to date provides a *bound* (hence the use of this word in the title of the method) on the optimal objective value which we have used to terminate the branch at node 2. Solving the LP relaxation at node 7 gives an objective value of $5\frac{1}{2}$ which again is below the current bound allowing us to terminate this branch. Since there are no more waiting nodes the integer solution found at node 6 has been shown to be optimal. This is the solution already obtained from Example 2.7 and given in (2.53).

Although we have obtained the optimal integer solution there is no guarantee that there might not be other (suboptimal) integer solutions if we had developed nodes 2 and 7. Exercise 2.6.8 involves developing these nodes to see if there are other integer (non-optimal) solutions.

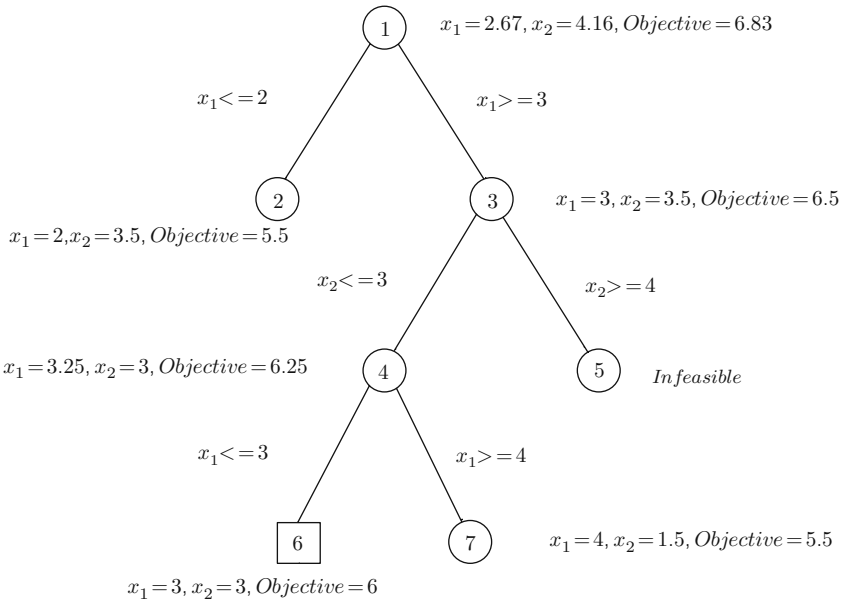

1
$x_1 = 2.67, x_2 = 4.16, Objective = 6.83$
$x_1 <= 2$
$x_1 >= 3$
2
3
$x_1 = 3, x_2 = 3.5, Objective = 6.5$
$x_1 = 2, x_2 = 3.5, Objective = 5.5$
$x_2 <= 3$
$x_2 >= 4$
$x_1 = 3.25, x_2 = 3, Objective = 6.25$
4
5
$Infeasible$
$x_1 <= 3$
$x_1 >= 4$
6
7
$x_1 = 4, x_2 = 1.5, Objective = 5.5$
$x_1 = 3, x_2 = 3, Objective = 6$

Fig. 2.6 A branch-and-bound tree

Exercise 2.6.9 is to resolve this model by branching on variable x_2 instead of x_1 at the top of the tree. This will result in a totally different tree but it should be apparent that it must also result in the same optimal objective value and, in this example, the same (unique) optimal solution (although, in general, there could be alternate optimal solutions).

A feature of this method, which should be remarked on, is that immediately after a branch the branching variable takes the value in the added constraint. This will always happen but there is no reason why the variable will not deviate from this value further down the tree, as for example happens at node 4. Once an extra constraint has been imposed at a branch this extra constraint will apply at all branches below the relevant node. For example x_1 is greater than or equal to 3 at all nodes from node 3 and below.

If an integer variable is restricted to a finite range of values (as is normally the case in practice) then it should be apparent that this method must converge. In the worse case, the branching process will result in **fixing** the integer variables at all the (finite number of) possible values at different nodes. This could result in an *exponential* number of nodes. The potential number of nodes at each successive level doubles giving 2^n nodes at level n. Exercise 2.6.10 demonstrates this. However, if some of the integer variables are not restricted to lie in a finite range then there is no guarantee that the method will terminate. Example 2.9 demonstrates this. This shows that branch-and-bound is not formally a decision procedure for IP in contrast to the method illustrated in Example 2.7.

Notice (for example) that the right branch from node 1 ($x_1 \geqslant 3$) and the left branch from node 4 ($x_1 \leqslant 3$) have fixed x_1 at 3.

It is useful to illustrate the branch-and-bound method, as performed in Fig. 2.6, geometrically in Fig. 2.7.

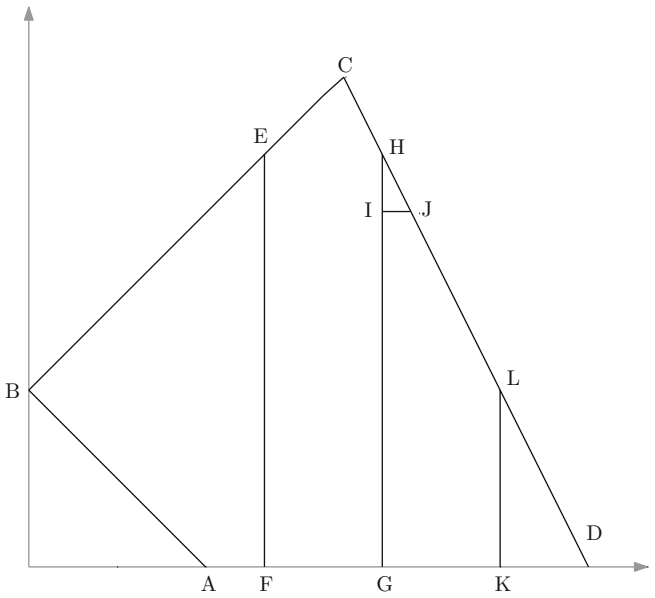


Fig. 2.7 A geometrical representation of branch-and-bound

ABCD represents the feasible region of the LP relaxation. The optimal solution of the LP relaxation corresponds to vertex C. By branching on x_1 we create the two regions ABEF and GHD corresponding to the LP relaxations at nodes 2 and 3. Vertices E and H correspond to the solutions at those nodes. Branching on x_2 at node 3 creates an empty feasible region for $x_2 \geqslant 4$ and the region GIJD for $x_2 \leqslant 3$. The solution at node 4 corresponds to vertex J. Branching on x_1 creates the feasible regions GI (one dimensional) and KLD. The solutions at nodes 6 and 7 correspond to vertices I and L. I clearly corresponds to the optimal integer solution.

Example 2.9 Demonstrate that branch-and-bound does not terminate with the following model:

Minimise

$$x_2 \tag{2.54}$$

subject to

$$3x_1 - 3x_2 \geqslant 1 \tag{2.55}$$

$$4x_1 - 4x_2 \leqslant 3 \tag{2.56}$$

$$x_1 \geqslant 1 \tag{2.57}$$

$$x_1, x_2 \in \mathbb{Z}$$

The resulting branch-and-bound tree is given in Fig. 2.8. It should be apparent that the tree will never terminate producing LP relaxations with ever-increasing objective values. Figure 2.9 demonstrates what is happening. The LP relaxation at node 1 corresponds to point A. After branching on x_2 we have alternate (all fractional) solutions to the resulting LP relaxation and choose (arbitrarily) that corresponding to point B. Continuing in this manner we produce solutions at points C,D,.... The problem is that the IP has no integer solution (it is infeasible). But branch-and-bound cannot detect this as it keeps finding feasible solutions to the LP relaxations. As mentioned before, in practice, this is unlikely to happen since integer variables usually have finite bounds. But it can be the case that branch-and-bound will take an exorbitant amount of time with some models. Exercise 2.6.11 is to solve this model by the decision procedure demonstrated in Example 2.7. This, of course, terminates finitely showing that there is no solution.

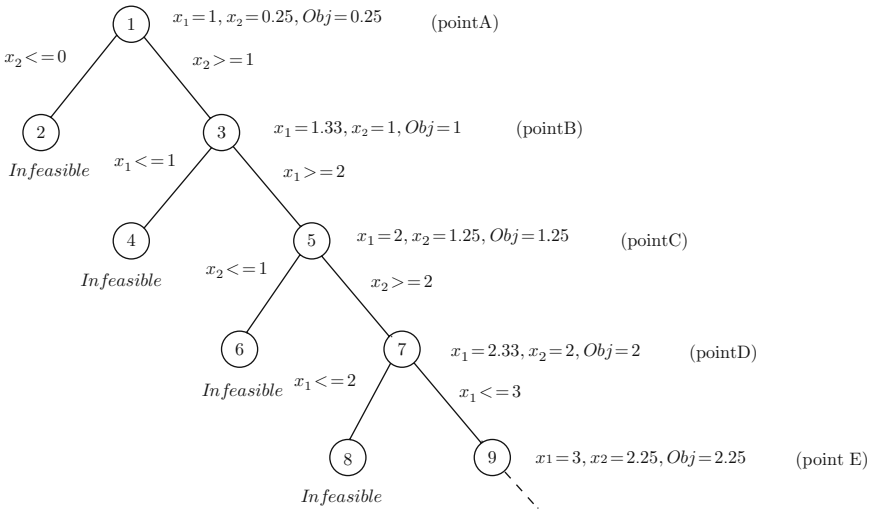


Fig. 2.8 A non-terminating branch-and-bound tree

If we apply branch-and-bound to a mixed IP then we only branch on those variables restricted to take integer values. If the integer variables are restricted to take the values 0 or 1 (in both the pure and mixed cases), as is most common, then the branch-and-bound tree becomes simpler. Each branch fixes a variable at 0 or 1 in the next and all subsequent nodes down the branch.

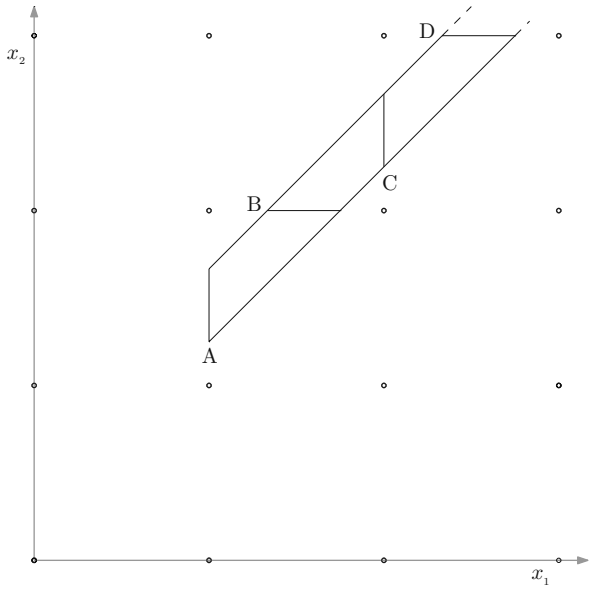


Fig. 2.9 Geometric representation of non-terminating branch-and-bound tree

2.2.2 The Convex Hull of an IP

In Fig. 2.10 we draw the feasible region of the LP relaxation of Example 2.7 (with the same lettering as in Fig. 2.7).

We also mark the points with integer coordinates inside this region. These are the feasible solutions to the IP in Example 2.7. Although the feasible solutions are disconnected, if we are using LP to help us solve the IP, we can surround these integer points by the smallest *convex region* containing them. This is known as the *convex hull* of feasible integer points. It is marked by the dotted lines in the figure. It is obvious that its boundaries must be straight lines in this two-dimensional case. In higher dimensions they will be hyperplanes. They could therefore be represented by linear constraints. In this example these are

$$x_1 + x_2 \geqslant 2 \tag{2.58}$$

$$x_1 \geqslant 1 \tag{2.59}$$

$$-x_1 + x_2 \leqslant 1 \tag{2.60}$$

$$x_2 \leqslant 3 \tag{2.61}$$

$$2x_1 + x_2 \leqslant 9 \tag{2.62}$$

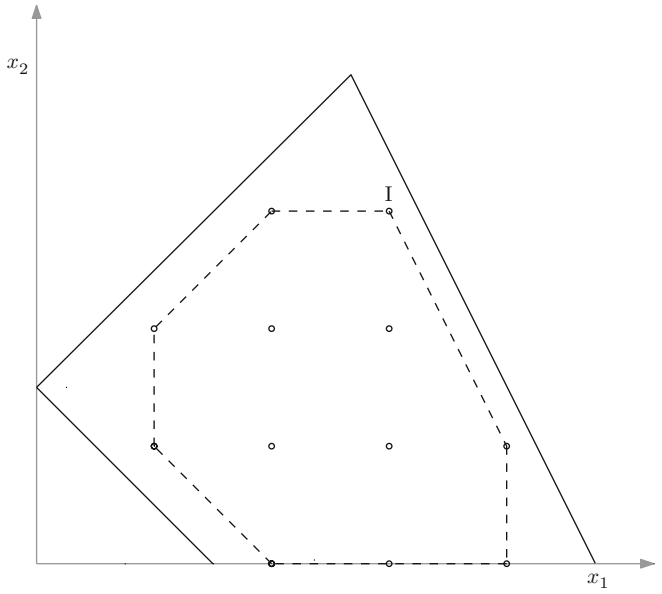


Fig. 2.10 The convex hull of a pure IP

$$x_1 \leqslant 4 \tag{2.63}$$

$$x_2 \geqslant 0 \tag{2.64}$$

If we were to use objective (2.41) and replace (or append) the constraints (2.42)–(2.45) by those above then the IP in Example 2.7 could be solved as a (much easier) LP. The optimal solution would be given by vertex I in Fig. 2.10.

The boundaries of the convex hull are known as *facets* and the constraints giving rise to them are known as *facet-defining* constraints.

In practice they may be very difficult to calculate. However, it may be possible to calculate some of them or approximate them by constraints which cut off some of the feasible points of the LP relaxation feasible region, but are not facets. All such constraints are known as *cutting planes.* Appending them to a model, when known, could be expected to reduce the size of the tree search in branch-and-bound. However, another problem, which frequently arises, is that there may be an astronomic number of facets. This is the case with, for example, all known IP formulations of the famous travelling salesman problem (TSP) (which is discussed in Sect. 2.3). In practice one may only add them (or other cutting planes) iteratively when they are known to cut off the current fractional solution.

Formulations that lead to constraints representing the convex hull are known as '*sharp*' and those which are closer to the convex hull than others are known as '*tighter*'.

Such procedures can be implemented as another method of solving IP models, i.e. solve the LP relaxation, add cutting plane(s) to remove the fractional solution

and keep repeating. Alternatively this approach can be implemented in a branch-and-bound tree (known as *branch-and-cut)* by applying it to the IPs corresponding to intermediate nodes in the solution tree.

If a model is a mixed IP then one is only restricted to integer coordinates in some directions. If, for example, we were only to restrict x_1 to take integer values in Example 2.7 the feasible solutions would be represented by the vertical lines in Fig. 2.11. The convex hull would be as represented by the dotted lines, and the optimal solution lies at vertex H.

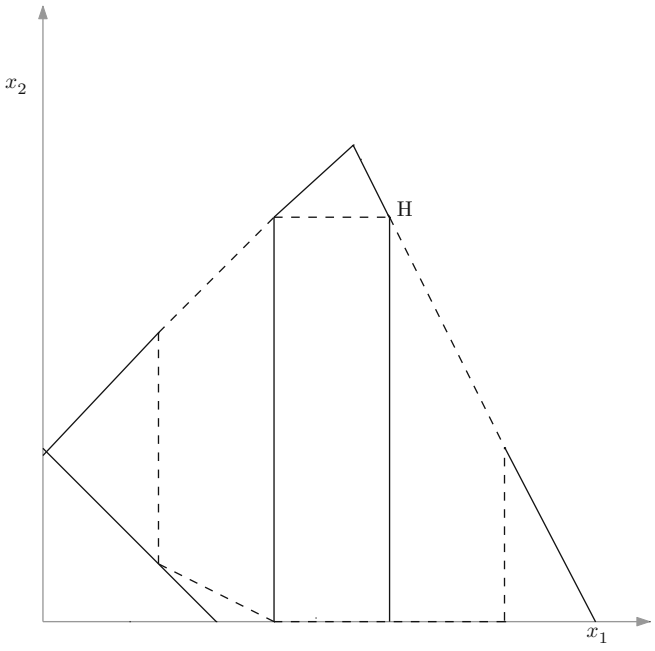


Fig. 2.11 Convex hull of a mixed integer programme

The facet-defining constraints for this example are then

$$x_1 + 2x_2 \geqslant 2 \tag{2.65}$$

$$x_1 \geqslant 1 \tag{2.66}$$

$$-2x_1 + 2x_2 \leqslant 3 \tag{2.67}$$

$$2x_2 \leqslant 7 \tag{2.68}$$

$$4x_1 + 2x_2 \leqslant 19 \tag{2.69}$$

$$x_1 \leqslant 4 \tag{2.70}$$

$$x_2 \geqslant 0 \tag{2.71}$$

The facet-defining constraints of an IP can be regarded as the strongest bounds on certain linear expressions which can be *inferred* from the constraints of an IP. This is analogous to the way in which LP can be regarded as an inference problem as mentioned in Sect. 1.1. Hence both types of problem could be regarded as inference problems. For example suppose we were to choose the objective

Maximise

$$x_2 \tag{2.72}$$

subject to the constraints of Example 2.7. The answer would clearly be $x_2 = 3$ as a result of the facet 2, i.e. the constraints *imply* $x_2 \leqslant 3$ but nothing stronger.

It should be obvious that it is not possible to derive the constraint $x_2 \leqslant 3$ by an argument involving only multipliers on the constraints, leading to a dual model. The best that can be obtained by such an argument is $x_2 \leqslant 4\frac{1}{6}$. $4\frac{1}{6}$ is the optimal objective value to the LP relaxation with objective (2.72). Viewed as inference problems finding the strongest implied constraints of an IP is therefore more difficult than simply adding constraints in certain multiples (the LP dual values). The difference between the optimal LP objective value and the optimal IP objective value is sometimes known as the *duality gap* for an IP because the LP optimal value is the best that can be obtained by dual multipliers. It is an indication of the difficulty of an IP. Clearly it is the difference between the objective value for the LP relaxation at the top of a branch-and-bound tree and the objective value at the node giving the optimal IP solution. Therefore it is some (albeit crude) measure of the likely size of the tree.

There is, however, a remarkable procedure for finding the strongest implied constraints (the facet-defining constraints) of a pure IP. It consists of the repeated application of the following operations. (For simplicity we will consider the case of a maximisation model with '$\leqslant$' constraints.)

i. Adding constraints together in suitable non-negative multiples.
ii. Dividing through by the greatest common divisor of the left-hand-side coefficients and rounding down the right-hand-side coefficient.

We illustrate this by reference to the constraints of Example 2.7.

Dividing constraints (2.43) and (2.44) each through by 2 we obtain, respectively

$$-x_1 + x_2 \leqslant \frac{3}{2} \tag{2.73}$$

$$2x_1 + x_2 \leqslant \frac{19}{2} \tag{2.74}$$

These imply, since the left-hand expressions are integer,

$$-x_1 + x_2 \leqslant 1 \tag{2.75}$$

$$2x_1 + x_2 \leqslant 9 \tag{2.76}$$

Multiplying (2.75) by 2 and (2.76) by 1 and adding we obtain

$$3x_2 \leqslant 11 \tag{2.77}$$

Dividing by 3 we obtain

$$x_2 \leqslant \frac{11}{3} \tag{2.78}$$

which, with rounding, implies

$$x_2 \leqslant 3 \tag{2.79}$$

giving the facet-defining constraint.

Unfortunately it is difficult to find a general way of carrying out this process systematically. Roundings are nested within the combining of constraints to an indeterminate depth. The depth of rounding is significant and is known as the *Chvátal rank.* Constraint (2.79) has Chvátal rank 2 since we had to apply two roundings to obtain it.

Since the strongest bound on any objective function for an IP (such as (2.41)) can be obtained as an appropriate linear combination of the facet-defining constraints (using LP dual multipliers) we can also obtain this using a nested combination of rounding and linear combinations of the original constraints. The depth of the rounding in this case is referred to as the Chvátal rank of the model. This is often used as a measure of its difficulty. Most of the 'easy' problems of IP have rank 0 (i.e. they can be solved as LPs) or 1. However, most well-known difficult classes of IP problems have 'unbounded' rank, i.e. one can find problems in the class which have a rank of any number one chooses. Again the rank of the TSP is unbounded.

It will be shown in Chapter 4 that one of the important methods of logical inference, i.e. *resolution,* can be represented in an IP form by rank 1 cutting planes.

Exercise 2.6.14 is to find the optimal bound on the objective of Example 2.7 as a *Chvátal function*, i.e. a nested combination of rounding and linear combinations.

In view of the importance of the special case of 0–1 IP models we illustrate the concept of the convex hull of a (small) pure 0–1 model.

Example 2.10 Find the convex hull of the feasible solutions to the following constraints:

$$\delta_1 + \delta_2 + 2\delta_3 \leqslant 2 \tag{2.80}$$

$$0 \leqslant \delta_1, \delta_2, \delta_3 \leqslant 1 \tag{2.81}$$

$$\delta_1, \delta_2, \delta_3 \epsilon \mathbb{Z}$$

Figure 2.12 illustrates the polytope associated with the LP relaxation.

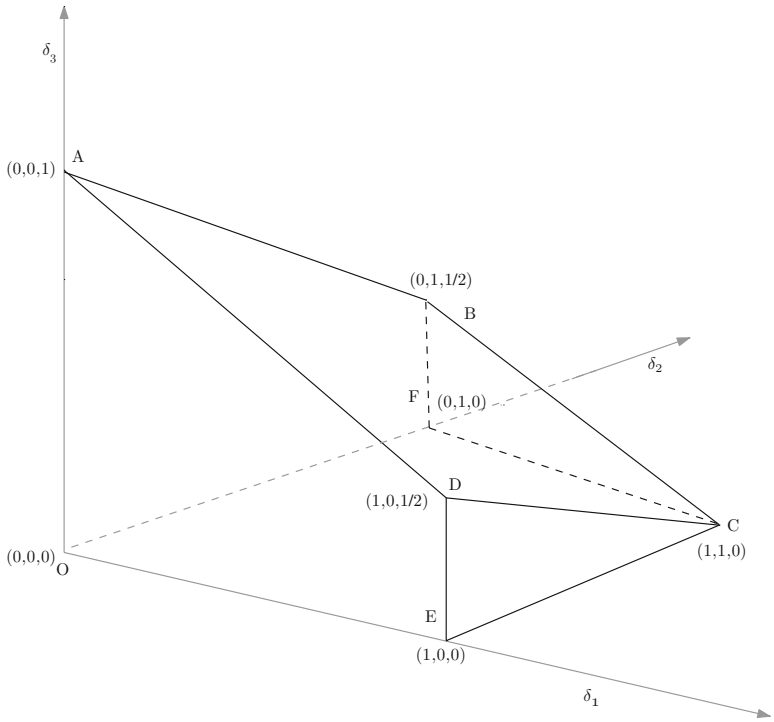


Fig. 2.12 The LP relaxation of a 0–1 IP model

Constraint (2.80) is represented by the plane ABCD. It can be seen that the polytope of the LP relaxation has vertices O,A,B,C,D,E and F. Hence optimising an objective subject to (2.80) and (2.81) (ignoring the integrality requirement) could lead to one of the solutions associated with the fractional vertices at B or D.

The convex hull of feasible 0–1 integer solutions is shown in Fig. 2.13.

The convex hull of the feasible integer solutions is defined by the constraints

$$\delta_1 + \delta_3 \leqslant 1 \tag{2.82}$$

$$\delta_2 + \delta_3 \leqslant 1 \tag{2.83}$$

together with (2.81).

It can be seen that all the vertices have 0–1 coordinates.

In general the feasible region associated with both the LP relaxation and the convex hull of a 0–1 IP model is a multidimensional hypercube with some of the corners sliced off.

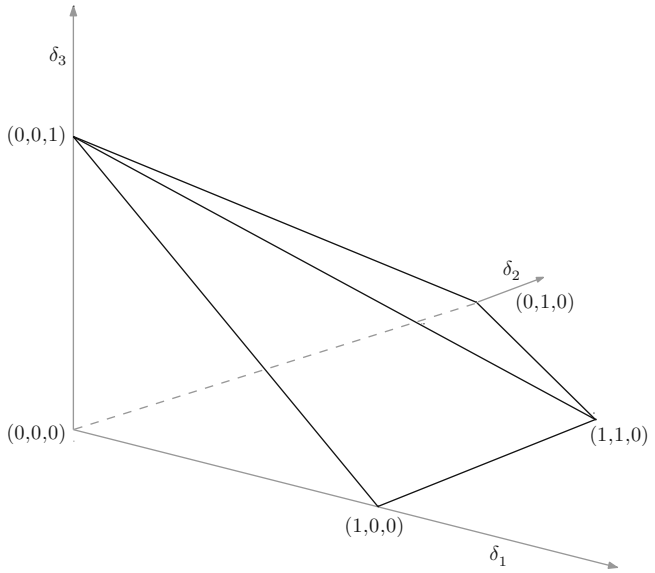


Fig. 2.13 The convex hull of a 0–1 IP model

2.3 The Use of 0–1 Variables

As has already been stated the majority of integer variables in practical IP models are restricted to the two values 0 and 1. If there are general integer variables then they would be restricted to a small integer range in almost any sensible IP model. Variables that, in reality, should be integer (e.g. number of cars manufactured or people of a certain category), which could take large values, would be treated as continuous and their solution values in the optimal solution rounded to give sensible answers.

2.3.1 Expressing General Integer Variables as 0–1 Variables

It should be pointed out that general (bounded) integer variables can always be expressed as a combination of 0–1 variables. We demonstrate by the following example.

Example 2.11 Express the integer variable x ($0 \leqslant x \leqslant U$) using 0–1 variables.

The most compact way of doing this is a 'binary' expansion

$$x = x_1 + 2x_2 + 4x_3 + 8x_4 + \cdots + 2^{\lfloor \log_2 U \rfloor} x_{\lfloor \log_2 U \rfloor} + 1 \tag{2.84}$$

where $x_1, x_2, \ldots, x_{\lfloor \log_2 U \rfloor} + 1$ are 0–1 variables.

Clearly by considering all possible 0 or 1 settings for these values we can represent any integer less than $2^{\lfloor \log_2 U \rfloor + 1}$. If necessary we may also impose the bound

$$x_1 + 2x_2 + 4x_3 + 8x_4 + \cdots + 2^{\lfloor \log_2 U \rfloor} x_{\lfloor \log_2 U \rfloor} + 1 \leqslant U \tag{2.85}$$

2.3.2 Yes/No Decisions

The 0–1 variables are usually used to represent yes/no decisions which by their nature must be *'discrete'*, i.e. 'do we build this warehouse here or not ?', 'do we make any of this product line or not ?', etc. Other decisions will be contingent on these decisions, hence the need for 'logical modelling'. We illustrate this by a number of examples. Systematic ways of modelling with 0–1 variables using logic are discussed in Chapter 3. Our purpose here is to show the modelling potential of IP and present some of the most common applications.

We cannot do justice to this topic in the short space of one section but we aim to give a taste for the modelling potential and sketch the major application areas. Further references are given in Sect. 2.5.

2.3.3 The Facility Location Problem

Example 2.12 Formulate the following ('capacitated') facility location problem as an IP.

We can build warehouses at locations $i \epsilon I$ to supply customers (with some commodity) at locations $j \epsilon J$. Over the period considered (say a month) a warehouse at i has capacity K_i and the customer at j has requirement D_j. The monthly *fixed* cost of a warehouse at i is f_i. The cost of supplying a unit of the commodity from the warehouse at i (if built) to the customer at j is c_{ij}. Choose where to build the warehouses and how much of the commodity to supply from each of them to each customer so as to minimise total cost.

We can introduce the following variables:

x_{ij} = monthly quantity sent from warehouse i to customer j.

$\delta_i = 1$ if a warehouse is built at i.

$= 0$ otherwise.

Our mixed IP model is then
Minimise

$$\sum_{i \epsilon I, j \epsilon J} c_{ij} x_{ij} + \sum_{i \epsilon I} f_i \delta_i \tag{2.86}$$

subject to

$$\sum_{j \epsilon J} x_{ij} \leqslant K_i \delta_i \quad \forall i \epsilon I \tag{2.87}$$

$$\sum_{i \epsilon I} x_{ij} = D_j \quad \forall j \epsilon J \tag{2.88}$$

$$x_{ij} \geqslant 0, \quad \forall i \epsilon I, \quad \forall j \epsilon J, \quad 0 \leqslant \delta_i \leqslant 1 \quad \forall i \epsilon I \tag{2.89}$$

$$x_{ij} \epsilon \mathbb{R}, \quad \delta_i \epsilon \mathbb{Z} \quad \forall i \epsilon I, \forall j \epsilon J$$

Notice this use of the quantifier '∀', but here it is $indices$ which are being quantified over the domains of their index sets. Such a notation is common in certain modelling languages as discussed in Chapter 3. The use of the Greek letter 'δ' for 0–1 variables is common in contrast to the Roman alphabet for continuous variables.

It is necessary to model this problem as a (mixed) IP because of the discrete nature of the decision to build, or not build, a warehouse at each location. The building of a fraction of a warehouse (resulting in an LP model) is not an option. The f_i are *fixed* costs which will either be incurred in their entirety, or not at all. These are in contrast to the c_{ij} which are *variable* costs which can be incurred in continuously varying fractions depending on the continuously varying values of the variables x_{ij}.

In (2.89) we have represented the variables δ_i as general integer variables which must lie in the range 0–1, so restricting them to the two values 0 and 1. It is more usual to specify them directly as *binary* variables $\delta_i \epsilon \{0, 1\}$.

This problem is an instance of the *fixed charge problem,* which is further discussed in Chapter 3. Note that it is not necessary to explicitly model the condition that if a warehouse is opened (and therefore a fixed cost incurred) one will send a positive quantity out of it. This must happen by virtue of optimality for to open it and not send any quantity from it would be non-optimal. Hence this condition is guaranteed by the objective function.

2.3.4 Logical Decisions

Once we have 0–1 variables we can make logical statements about the corresponding decisions. Example 2.13 illustrates this.

Example 2.13 Model the following extra conditions in the facility location problem.

i. At most n warehouses can be built where $n \leqslant |I|$.
ii. If any warehouses are built at $i \epsilon P$ then no warehouses can be built at $k \epsilon Q$ but at least three warehouses must be built at $l \epsilon L$.

Condition (i) can be modelled by

$$\sum_{i \epsilon I} \delta_i \leqslant n \tag{2.90}$$

Condition (ii) can be modelled by

$$\delta_i + \delta_k \leqslant 1 \quad \forall i \epsilon P, \quad \forall k \epsilon Q \tag{2.91}$$

$$\sum_{l \epsilon L} \delta_l \geqslant 3\delta_i \quad \forall i \epsilon P \tag{2.92}$$

Many variants and extensions of the facility location problem can also be modelled. The problem typifies a whole class of problems involving discrete decisions which have to be made if other activities are to be carried out.

Example 2.14 A blending problem has been formulated as an LP with variables x_i representing the proportions of ingredients i. In addition it is stipulated that

i. If an ingredient i is included it must be at a proportion of at least α_i.
ii. At most n ingredients can be used.
iii. If any ingredient $i \epsilon I$ is used then at least one ingredient $j \epsilon J$ must also be used.

We introduce 0–1 variables δ_i representing the presence of ingredient i and the following constraints:

$$x_i \leqslant \delta_i \tag{2.93}$$

$$x_i \geqslant \alpha_i \delta_i \tag{2.94}$$

If $x_i > 0$ then (2.93) forces $\delta_i = 1$. Then $\delta_i = 1$ forces $x_i \geqslant \alpha_i$ by (2.94).
Condition (ii) is similar to that in Example 2.13 and is modelled as

$$\sum_i \delta_i \leqslant n \tag{2.95}$$

Condition (iii) is modelled as

$$\sum_{j \epsilon J} \delta_j - \delta_i \geqslant 0 \quad \forall i \epsilon I \tag{2.96}$$

Again the above types of constraints may also arise in other contexts than blending.

It is sometimes natural to model certain conditions as products of 0–1 variables. Of course such a product is itself a 0–1 quantity which can be represented as a 0–1 variable with a logical dependence on the original variables. Example 2.15 illustrates this.

2.3.5 Products of 0–1 Variables

Example 2.15 A model contains the product $\delta_1\delta_2$ of two 0–1 variables. Convert the model to a linear form.

We replace this product with a new 0–1 variable γ and the constraints

$$\delta_1 + \delta_2 - \gamma \leqslant 1 \tag{2.97}$$

$$\delta_1 - \gamma \geqslant 0 \tag{2.98}$$

$$\delta_2 - \gamma \geqslant 0 \tag{2.99}$$

Equation (2.97) forces γ to be 1 if δ_1 and δ_2 are both 1. Equations (2.98) and (2.99) force γ to be 0 if either δ_1 or δ_2 (or both) take the value 0.

2.3.6 Set-Covering, Packing and Partitioning Problems

There is an important class of problems which give rise to pure 0–1 IP models known as *set-covering, packing* and *partitioning* models. We illustrate this by an example of a set-covering model.

Example 2.16 Cover all the elements {1,2,3,4,5} using the minimum number of subsets {1,2,3,5}, {1,2,4,5}, {1,3,4}, {2,3,4,5}, {3,4,5}.

We use 0–1 variables δ_j which equal 1 if and only if the jth subset is used. Each element of the original set gives rise to a constraint. In order to cover an element the sum of the variables representing the subsets containing the element must be at least 1. The model then becomes

Minimise

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 \tag{2.100}$$

subject to

$$\delta_1 + \delta_2 + \delta_3 \geqslant 1 \tag{2.101}$$

$$\delta_1 + \delta_2 + \delta_4 \geqslant 1 \tag{2.102}$$

$$\delta_1 + \delta_3 + \delta_4 + \delta_5 \geqslant 1 \tag{2.103}$$

$$\delta_2 + \delta_3 + \delta_4 + \delta_5 \geqslant 1 \tag{2.104}$$

$$\delta_1 + \delta_2 + \delta_4 + \delta_5 \geqslant 1 \tag{2.105}$$

$$\delta_1, \delta_2, \delta_3, \delta_4, \delta_5 \epsilon \{0, 1\}$$

This type of model arises in a number of contexts. One of the most important is crew scheduling (particularly for airlines) where each subset corresponds to a roster which, if used, must be assigned to a possible crew. The objective is to minimise the number of crews necessary to cover all the flights. Each flight gives rise to a constraint. The model also arises in computer circuit design and logic simplification. This latter application is discussed in Chapter 4.

It is worth pointing out that, in practice such models can have a huge number of variables. However, they are often amenable to simplification by *preprocessing*. For example δ_5 is redundant since δ_4 covers everything that δ_5 does and more at equal objective 'cost'. The fifth constraint can then be removed as it is implied by the second. This reduces the model to

Minimise

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 \tag{2.106}$$

subject to

$$\delta_1 + \delta_2 + \delta_3 \geqslant 1 \tag{2.107}$$

$$\delta_1 + \delta_2 + \delta_4 \geqslant 1 \tag{2.108}$$

$$\delta_1 + \delta_3 + \delta_4 \geqslant 1 \tag{2.109}$$

$$\begin{gathered} \delta_2 + \delta_3 + \delta_4 \geqslant 1 \\ \delta_1, \delta_2, \delta_3, \delta_4 \in \{0, 1\} \end{gathered} \tag{2.110}$$

Exercise 2.6.17 is to solve this model by branch-and-bound.

Set-covering models look deceptively easy to solve but can be very difficult, usually on account of their size. They do have important mathematical properties. In particular there will always be an optimal integer solution which corresponds to a vertex of the polytope for the LP relaxation (but normally not the vertex associated with the LP optimum). The Chvátal rank for the class of set-covering models is not bounded.

Closely related are set packing models where the objective is to 'pack' as many subsets into a given set, as possible, with no overlap. A particularly important case arises where the set of elements can be represented as the nodes of a graph. The subsets are pairs of nodes which are joined by an edge, The packing problem becomes that of finding the maximum 'pairing' of nodes where each node has a unique 'mate' (if it has one at all). This is known as the *matching problem.* Formulated as an IP it can be shown to be of Chvátal rank 1. A special case of the matching problem is to 'bipartite' graphs. Here there are two sets of nodes and matchings can only be between nodes of different sets. This problem is usually known as the *assignment* problem. An optimal solution to the LP relaxation is integer. The Chvátal rank is therefore 0. A set partitioning model is that of partitioning the original elements into a collection of the given subsets. Packing and partitioning models can be converted

into one another. In all these types of models, with some applications, objectives with non-unit coefficients may be used.

2.3.7 Non-linear Problems

Example 2.17 demonstrates how a certain type of non-linearity in an IP model may be linearised. Non-linear models in general may be remodelled, to a degree of approximation and an increase in size, as linear IP models. Hence the distinction between linear and non-linear in IP is less important than in other branches of mathematics. What's more the IP formulation has the advantage that *global* optimal solutions are obtained in contrast to *local* optimal solutions. This distinction is illustrated in Fig. 2.14.

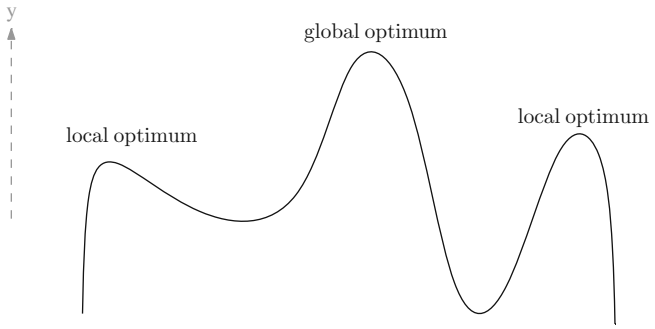


Fig. 2.14 Local and global optima

If our objective were to maximise y then conventional optimising methods might find one of the local optima shown and not be able to improve on these. However, an IP formulation can be guaranteed to find a global optimum (at, possibly, considerable computational cost). The next example illustrates how non-linear models may be formulated as linear IP models.

Example 2.17 Make a piecewise linear approximation to the non-linear function $y = x^3 - 12x^2 + 23x + 86$ for $0 \leqslant x \leqslant 39$ and model the (approximate) relationship between x and y using an IP model.

Such an expression (together with other non-linear expressions) may occur in the objective, the constraints or both.

Figure 2.15 gives the graph of $y = x^3 - 12x^2 + 23x + 86$.

We evaluate the function at a number of *grid points*. These need not necessarily be evenly spaced, but for this example we evaluate the function at integers between $x = 0$ and 9 giving Table 2.1.

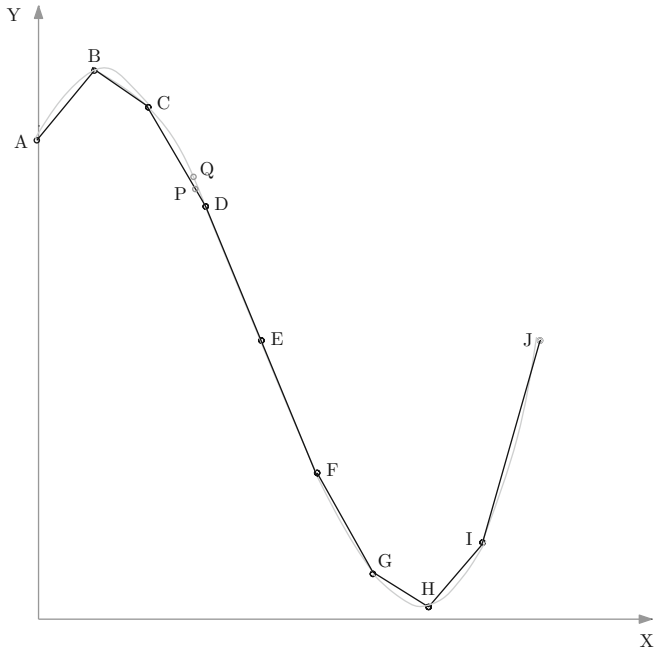


Fig. 2.15 A piecewise linear approximation to a non-linear function

Table 2.1 Values for a Non-linear Function

x	0	1	2	3	4	5	6	7	8	9
y	86	98	92	74	50	26	8	2	14	50

Joining these points in Fig. 2.15 gives the piecewise linear approximation ABCDEFGHIJ to the curve for the function. In order to do this we introduce (non-negative continuous) variables $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8, \lambda_9$ which represent 'weights' given to the points A,B,C,D,E,F,G,H,I,J in representing the function by the following equations:

$$x = 0\lambda_0 + 1\lambda_1 + 2\lambda_2 + 3\lambda_3 + 4\lambda_4 + 5\lambda_5 + 6\lambda_6 + 7\lambda_7 + 8\lambda_8 + 9\lambda_9 \tag{2.111}$$

$$y = 86\lambda_0 + 98\lambda_1 + 92\lambda_2 + 74\lambda_3 + 50\lambda_4 + 26\lambda_5 + 8\lambda_6 + 2\lambda_7 + 14\lambda_8 + 50\lambda_9 \tag{2.112}$$

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 + \lambda_9 = 1 \tag{2.113}$$

$$\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8, \lambda_9 \geqslant 0 \tag{2.114}$$

$$\lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8, \lambda_9 \epsilon \mathbb{R}$$

In addition we need to impose the condition

$$\text{At most 2 adjacent } \lambda_i \text{ can be non-zero} \tag{2.115}$$

If λ_i and λ_{i+1} are the two adjacent non-zero λ_i's then (x, y) represents a point on the line between the points corresponding to the ith and $(i + 1)$th grid points. For example if $\lambda_0 = 0, \lambda_1 = 0, \lambda_2 = \frac{1}{4}, \lambda_3 = \frac{3}{4}, \lambda_4, \ldots, \lambda_9 = 0$ then $(x, y) = (2\frac{3}{4}, 78\frac{1}{2})$ being the point P in Fig. 2.15. On the true curve if $x = 2\frac{3}{4}$ then $y = 79\frac{19}{64}$ being the point Q. The piecewise linear approximation has therefore underestimated the true value of y by $\frac{51}{64}$. Obviously the more refined the grid the more accurate the approximation, but more λ_i variables will be needed.

It remains to represent condition (2.115) by constraints. In order to do this integer variables are needed. We need to model the logical condition that the possible non-zero pair is either λ_0 and λ_1, or λ_1 and λ_2, or λ_3 and λ_4, etc. This can be done by 0–1 variables $\delta_1, \delta_2, \ldots, \delta_9$ and the constraints

$$\lambda_0 \leqslant \delta_1 \tag{2.116}$$

$$\lambda_1 \leqslant \delta_1 + \delta_2 \tag{2.117}$$

$$\lambda_2 \leqslant \delta_2 + \delta_3 \tag{2.118}$$

$$\vdots$$

$$\lambda_8 \leqslant \delta_8 + \delta_9 \tag{2.119}$$

$$\lambda_9 \leqslant \delta_9 \tag{2.120}$$

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_7 + \delta_8 + \delta_9 = 1 \tag{2.121}$$

Constraint (2.121) ensures that exactly one δ_i is 1, the others being 0. Then by constraints (2.116)–(2.120) at most two adjacent λ_i can be non-zero.

In order to 'linearise' expressions such as that above in this way they must be functions of a single variable or be able to be expressed as the sum of non-linear functions of single variables. Such functions are known as '*separable.*' The limitation of only being able to model separable functions is not as restrictive as might, at first, be thought. For most functions arising in practice it is usually possible to 'separate' them by suitable transformations. For example the product xy of two continuous variables, although not separable, can be transformed into a separable model by the log transformation since $\log xy = \log x + \log y$.

We again discuss how non-linear problems give rise to IP models and the important distinction between *convex* and *non-convex* models in Chapter 3.

2.3.8 The Knapsack Problem

This is one of the simplest types of pure IP model involving only one constraint and takes the form

Maximise

$$\sum_j c_j x_j \tag{2.122}$$

subject to

$$\sum_j a_j x_j \leqslant b \tag{2.123}$$

$$x_j \geqslant 0 \quad \forall j \tag{2.124}$$

$$x_j \epsilon \mathbb{Z} \quad \forall j$$

Like some of the other IP models discussed in Sect. 2.4, under the title of 'computational complexity', this type of model is 'difficult' to solve but, in one sense, the easiest of such IP models. Extensions of the knapsack problem are the bounded variable knapsack problem, the 0–1 knapsack problem and the equality-constrained knapsack problem. The name arises from the largely fictitious application of trying to fill a knapsack of limited size with items of maximal total value (although it has been used to allocate the hardware for different experiments in space exploration vehicles). More commonly it is used as a subproblem for other combinatorial problems (e.g. cutting stock and bandwidth allocation).

The LP relaxation of the knapsack problem is trivial to solve (concentrate on the activity with the greatest ratio of objective coefficient/constraint coefficient) but the IP involves finding the best combination of items to fill the knapsack taking account of both this ratio and how well it fits (with other items) into the knapsack.

2.3.9 The Travelling Salesman Problem

We demonstrate this problem by the following example.

Example 2.18 A routing problem. Find the shortest order to go around cities $N = \{1, 2, \ldots, n\}$, returning to the beginning. The distance from city $i \rightarrow j$ is given as c_{ij}.

We will assume that the distance from i to j is not, necessarily, the same as the distance from j to i, i.e. we have an example of the *asymmetric travelling salesman problem* (TSP).

We can model this problem as

Minimise

$$\sum_{i,j} c_{ij} x_{ij} \tag{2.125}$$

subject to

$$\sum_{j} x_{ij} = 1 \quad \forall i \epsilon N \tag{2.126}$$

$$\sum_{i} x_{ij} = 1 \quad \forall j \epsilon N \tag{2.127}$$

$$\sum_{i,j \epsilon S} x_{ij} \leqslant |S| - 1 \quad \forall S \subset N \text{ such that } 1 < |S| < n \tag{2.128}$$

$$x_{ij} \epsilon \{0, 1\} \quad \forall i, j \epsilon N$$

The 0–1 variables x_{ij} indicate whether a direct link $i \rightarrow j$ is, or is not, on the tour. Constraint (2.126) forces each city to be left exactly once and constraint (2.127) forces each city to be entered exactly once. Constraint (2.128) rules out *subtours*. Figure 2.16 demonstrates subtours which would be ruled out by constraints (2.128). Note that there are $2^n - n - 1$ subtour elimination constraints, i.e. an exponential number as a function of n. In practice these constraints are only added on an 'as needed basis', or known facet constraints are added when they cut off fractional solutions to the LP relaxation.

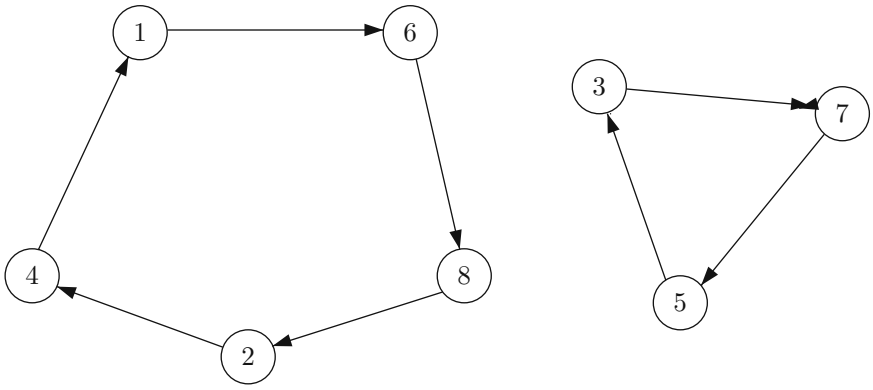


Fig. 2.16 Subtours of a TSP

There are other formulations of the TSP, as an IP, with a polynomial number of constraints although they have weaker LP relaxations (with one exception). References are given in Sect. 2.5.

2.3.10 Other Problems

In addition to the applications of 0–1 IP discussed above there are a number of other applications of 0–1 IP not discussed here but which can be followed up through the references.

Other problems involve networks and graphs (with nodes and arcs) and are naturally modelled with 0–1 variables representing the presence of particular nodes and the resulting presence of arcs emanating from them. The *quadratic assignment problem* is an extension of the assignment problem mentioned above (which can be treated as an LP) but is naturally modelled with products of 0–1 variables which can be linearised (with a large increase in model size) in the way shown in Example 2.15 making it a genuine IP. The *matching problem* that was mentioned above can also be regarded as an extension of the assignment problem. The matching problem, itself, has a number of extensions.

Scheduling problems involve deciding in what order to process jobs on machines. The 0–1 variables can be used to decide, for each pair of competing jobs, which one is processed first.

Capital investment problems (in addition to facilities location, already discussed) can incorporate capital limitation constraints and be modelled over periods of time with resultant cash flows.

2.4 Computational Complexity

2.4.1 Problem Classes and Instances

It is clear that problems in some classes are more difficult to solve than others. For example LPs are relatively 'easy' to solve. Models with thousands of constraints and variables are now solved routinely on desktop computers. Models with millions of constraints and variables have also been successfully solved. For IP the picture is very different. While some large models have been successfully solved some models, with only hundreds of variables, can prove very difficult. In logic large satisfiability problems (which are discussed in Chapter 4) can prove very difficult to solve. The subject of computational complexity is concerned with trying to classify problems into the easy and difficult categories. According to the method which we will describe LP is easy but IP, and many of its special cases, and the satisfiability problem are difficult.

The method which we describe looks at *classes* of problems rather than *instances.* Within a particular 'difficult' class there may be easy instances. The categorisation of classes will be based on the *worst cases* within that class. In some ways this is unsatisfactory since in practice the worst cases may seldom arise. However if we base the categorisation on worst cases then we can give a 'guarantee' of difficulty.

Average case analysis is much more difficult and requires probability distributions to be defined in order to define what the average cases are.

It is important to recognise that the method uses categorises problem classes rather than algorithms. There may be 'good' and 'bad' algorithms for solving problems in any class. For a problem class to be designated easy it is necessary to show that there is at least one easy algorithm for solving all problems in the class. A problem class will be designated 'easy' if there is an algorithm for solving problems in it whose number of steps will never be more than a *polynomial* function of the size of the problem (storage spaces required in a computer). On the other hand a problem class will be designated 'difficult' if there is no known such algorithm and the number of steps in all the algorithms known requires an *exponential* number of steps. These definitions will be made more precise later. A rather negative aspect of the categorisation, in terms of mirroring reality, is that some successful algorithms for problems in the easy class are exponential (in the worst case). The classic example is the simplex algorithm for LP. Examples can be found (see references) where the simplex algorithm takes an exponential number of steps although in practice ('average cases') it is known to solve huge models in reasonable periods of time. However there are known to be polynomial algorithms for LP (one of which sometimes overtakes the Simplex on 'huge' problems). For IP and the satisfiability problem no algorithms have ever been found which are not exponential or worse.

2.4.2 Computer Architectures and Data Structures

Another aim of this problem characterisation is to make it, as far as possible, independent of the computer architecture used. In order to do this fairly crude assumptions are made but it turns out that they often do not matter in making the (major) distinction between polynomial and exponential algorithms. One of the assumptions is that 'elementary' operations such as addition, multiplication, comparison, etc. all take the same amount of time on 'small' numbers. Of course multiplication is inherently more difficult than addition, although by how much depends on the architecture of the computer used.

Also of great significance, in practice, is the *data structure* used for encoding instances of the problem. We must, however, restrict ourselves to 'sensible' data structures. We would not encode a number n by a sequence of n 1's. We would, rather, encode it using binary notation taking account of the significance of the position of each digit as well as its value. For example 23 would be written as 10111 (since $23 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$) instead of as a string of 23 1's. It is useful (and realistic) to use only two symbols 0 and 1 given the binary hardware devices used in computers, although representing numbers to a higher radix would not alter the final analysis. However, what is important is to use some such representation. Clearly the storage required for a number n will be $\log_2 n$ bits (with possibly another bit to represent the sign of n), as opposed to n bits.

2.4.3 Polynomial and Exponential Algorithms

If the storage required for a problem instance is N bits then we will represent the number of elementary operations needed to solve it by $f(N)$. If $f(N) \leqslant cN^r$ for some algorithm and for **all** instances in the class, for a particular c and a particular r then the algorithm is said to *polynomially bounded* and the class of problems is said to belong to category $\mathbb{P}$ (polynomially solvable). We seek the algorithm with smallest value of r for which this inequality holds. This definition has certain limitations. First we do not normally take account of the magnitude of c. This means that, in practice, a very large number of operations might be needed **but** the growth in the number of operations needed, as problem sizes increase, will be less than for non-polynomially bounded classes. This measure is an *asymptotic* one, measuring rate of computational growth rather than computational difficulty for particular problem instances. One of the reasons for paying little attention to the value of c (and, to a lesser extent, r) is that these values can be altered by different implementations of an algorithm and the data structures used.

If the rate of growth of an algorithm cannot be shown to be polynomially bounded then we could look at gradually higher rates of growth, e.g. $N^{\log\log\ldots\log N},\ldots,N^{\log N}$. However, we do not usually examine these possible bounds on growth but look at exponential growth rates. If no algorithm, for a particular problem class, can be found which is polynomially bounded we see if we can find a bound of the form $f(N) \leqslant ca^N$. The number of computational steps is then said to be *exponentially bounded.* This does not mean that there does not exist a polynomially bounded algorithm for problems in the class but, rather, that one has not been found. There is, however, strong experimental and theoretical evidence that for many of the 'difficult' problems (e.g. IP and many special cases) we consider that no polynomially bounded algorithm can exist. Before we discuss why this is so it is worth illustrating the dramatic difference between polynomial and exponential growth by comparing the growth in r^2 operations with that of 2^r operations. This is done in Table 2.2 where we assume each elementary computer operation takes 1 ms. Then the computational times are those given.

Table 2.2 Exponential Versus Polynomial Growth

r	r^2	2^r
1	0.000001 s	0.000002 s
10	0.0001 s	0.001 s
20	0.0004 s	1 s
30	0.0009 s	17.9 min
40	0.0016 s	12.7 days
50	0.0025 s	35.7 years
60	0.0036 s	366 centuries

It is worth noting, at this point, why the encoding of the problem data was important. Suppose we had not used a binary encoding (or an encoding to some other radix), but simply a string of digits of length equal to the size of the data. Instead of

being of size N the data would have been of size $M = 2^N$ (since $\log_2(2^N) = N$). Then an exponential function of N, a^N would become $M^{\log_2 a}$, i.e. a polynomial function of M and the distinction between exponential and polynomial growth would be lost.

If, however, we were to encode to a different radix than 2 then this would simply alter the value of c for a polynomial growth rate and the value of a for an exponential growth rate. It would not, therefore, alter the distinction between polynomial and exponential growth rates.

We should also explain that, in this discussion, we are restricting ourselves to data and solutions that are integer or rational. Therefore these can be represented exactly in a computer and we do not have to bring in concepts of solution to some degree of approximation. Of course it is possible to represent any 'computable' number (i.e. any we might wish to talk about such as, for example, $\sqrt{2}$ or π) in a computer by an exact 'formula' (computer procedure) but such issues of a uniform method of representability are beyond the scope of this book. We restrict ourselves to the usual (binary) encoding of numbers to successive multiples of powers to a given radix. If necessary rational numbers could be represented exactly by storing their numerator and denominator (reduced to be coprime), it is easy to show that the size of the denominator is bounded by the maximum determinant of a basis matrix in LP.

It is necessary to distinguish between 'small' and 'large' numbers in discussing the encoding. While each 'number' in some problems (e.g. the satisfiability problem and some graph problems) can be represented as a small number and elementary operations on such numbers assumed to take unit time, this is unrealistic for other problems. For example LP and IP problems may contain arbitrarily large coefficients and the time to carry out operations on them will be related to (the logarithm of) their size. Therefore, for example, we could take the size of the encoding of an LP or IP to be bounded by $(m+1) \times (n+1) \times \log \max_{ij}(|A_{ij}|, |c_j|, |b_i|)$, where m, n, A, c, b represent the number of constraints, variables, matrix of coefficients and objective and right-hand-side vectors, respectively. Other ways of measuring the size of the encodings (e.g. $\log(\Sigma_{ij}(A_{ij} + c_j + b_i))$ will make no difference to whether they be classified as polynomial or exponential. With any of these encodings LP becomes polynomially bounded (by the Ellipsoid and Karmarkar algorithms referenced in Sect. 2.5). However no polynomial algorithm has ever been found (or seems likely to be found) which is a function of m and n alone, i.e. which ignores the magnitude of the coefficients. In contrast there are polynomial algorithms for solving simultaneous equations (e.g. by Gaussian elimination) which are functions of m and n alone. Another reason suggesting why it is usually necessary to take account of the magnitude of the coefficients is that it is, in principle, possible to reduce many problems to 'simpler' problems with larger coefficients. Any pure IP model, with all equality constraints, can be converted to a knapsack model by aggregating the constraints (but producing very large coefficients in consequence). In one sense the knapsack problem is 'easier' than general pure IP (although both are difficult in the classification given here). It would therefore be unrealistic to assume the size of the coefficients in a knapsack problem do not matter. The celebrated work of Gödel, mentioned in Chapter 1, was carried out (again in principle) by encoding proofs of

theorems uniquely as numbers and then defining relations between these numbers. If carried out in practice these 'Gödel numbers' would be astronomic. But they do demonstrate that any 'problem' can be represented by a number and the problem of solving it reduced to a computation on that number. The size of this number (a 'super-exponential' function of the numbers in the original problem) must, obviously, be realistically taken into account when discussing the complexity of solving the original problem. (What Gödel did was to create a number 'G' (in principle) which represented the formalised statement (in the meta system) 'The statement with Gödel number G is not provable'. Hence if G represented a 'true' statement (in the formalisation of the meta system) then it could not be proved within the system and vice versa. This also demonstrates that there are problems for which there is no bound on the number of computational steps needed. They are 'unsolvable'.

2.4.4 Non-deterministic Algorithms and Polynomial Reducibility

In order to analyse the computational complexity of the difficult problems which we consider in this book we note that the (optimal) solutions to many of them could be checked in polynomial time if we had a way of checking all the candidate solutions in parallel. This is a theoretical, but practically unrealisable notion. We call a 'machine' which can carry out such a check of all possible cases in parallel a *non-deterministic* computer. If each check is polynomially bounded then the algorithm is said to be *non-deterministically polynomially bounded.* If there exists such an algorithm for a class of problems the class is said to belong to the category $\mathbb{NP}$. It turns out that most of the problems which we consider in this book belong to $\mathbb{NP}$. Of course problems that are in $\mathbb{P}$ are also in $\mathbb{NP}$ since a problem that can solved in polynomial time can trivially have the solution checked in polynomial time (without even the need for a non-deterministic machine). However, to show that a problem class is not polynomially bounded we need to show that it is in $\mathbb{NP}$ but not in $\mathbb{P}$. It has never been proved that there are such classes, i.e. that $\mathbb{P} \neq \mathbb{NP}$. However, there is strong evidence to suggest it is true. We therefore seek to find the 'most difficult' problem classes in $\mathbb{NP}$. The problems here are referred to as '$\mathbb{NP}$ *complete*' and the category as $\mathbb{NPC}$. It turns out that all the known 'difficult' problem classes, which we are considering here, e.g. IP and various applications such as the TSP as well as the satisfiability problem, are '*polynomially reducible*' to each other. This means that if we take a problem P_1 in a 'difficult' category C_1, which is encoded by a string N_1, then we can transform it in a number of steps $g(N_1)$, which is polynomially bounded, into a (polynomial) encoding N_2 of a problem P_2 in a class C_2. If there is no polynomially bounded algorithm for the problems in C_1 then there can be no polynomially bounded algorithm for the problems in C_2. Otherwise we could use such an algorithm to construct a polynomially bounded algorithm for class C_1. In Chapter 4 we discuss the satisfiability problem and show that it can naturally be converted into a set-covering problem (a special case of IP). The transformation is immediate and, therefore trivially polynomial. Therefore if

there is no polynomially bounded algorithm for the satisfiability problem (and none has ever been found) there can be none for the set-covering problem or general IP. It should be pointed out that such polynomial reductions can be quite 'radical' as their only real purpose is theoretical. The reduced problems may bear little practical resemblance to the problems from which they are reduced. Historically the satisfiability problem was taken as class C_1. No polynomially bounded algorithm has ever been found for all problems in this class (although there are polynomially bounded algorithms for specialised cases). It can be transformed into other difficult problem classes and vice versa. Once we have a polynomial transformations between this and other problem classes all these classes can be put in $\mathbb{NP}$ and any transformation between another problem class and any class in $\mathbb{NP}$ places the new class in $\mathbb{NP}$. It might be thought that the reformulation of IP models, in terms of their convex hull, makes them polynomially solvable, as they become LPs. But, in general this will result in an exponential number of constraints (as a function of the problem size). Also the amount of calculation needed to produce facet-defining constraints may not be a polynomial function of the problem size.

There are also 'more difficult' problems than those in $\mathbb{NP}$, i.e. problems where there is no known non-deterministic polynomial-checking algorithm. Such problems have been named $\mathbb{NP}$ *hard*, i.e. they belong to the $\mathbb{NPC}$ class or are 'harder' than any problems in this class. While $\mathbb{NPC}$ problems may be polynomially reducible to such problems the reverse will not be the case. It is rather unfortunate that $\mathbb{NP}$ complete problems are sometimes referred to as 'intractable'. This is falsely taken, by some people, to imply that there is no point in seeking exact algorithms for them. Instead (polynomially bounded) heuristics must be sought to produce approximate solutions (possibly within some error bound). This is false. There are many problems belonging to 'difficult' classes which are reasonably easily solved for most practical instances. The existence of (sometimes pathological) worst cases which may take a long time to solve should not deter one from seeking exact solutions in practice.

2.4.5 Feasibility Versus Optimisation Problems

Some of the problems that we have discussed are optimisation problems (e.g. LP, IP and the many special cases) and some are feasibility problems (e.g. the satisfiability problem and certain decidability problems). It is usual, when doing complexity classifications, to regard all problems as feasibility problems. Suppose, for example, we have an IP model, which is a maximisation, and we know lower (l) and upper (u) bounds on the optimal objective value. We can test if z_0 is the optimal objective value by checking if the constraint $objective \geqslant z_0$, when appended to the constraints, keeps the model feasible but the constraint $objective \geqslant z_0 + 1$ makes it infeasible. The value of z_0 can be found, in a polynomial number of steps, by dissecting $u - l$ to test $z_1 = (u - l)/2$ (or nearest integer) for feasibility. If feasible the new interval for dissection is $z_1$to u. If infeasible the new interval is l to z_1.

Proceeding in this way the optimal solution can be found with $\log_2(u-l)$ feasibility tests. Hence polynomial tests for optimality imply polynomial tests for feasibility and vice versa.

2.4.6 Other Complexity Concepts

Our categorisation has been based on the *time* to carry out computer operations. There is also interest in the *space* required, e.g. what is the maximum space requirement as a function of the size of the input data? We will not cover such an analysis here. However, time and space complexity are connected since if we require an exponential number of storage positions we will require an exponential time to read them. However, the reverse need not be the case. We may require an exponential time but be able to carry out the calculation in a polynomial space.

Another indicator of the 'easiness' of some problems is an easy check for optimality. For example, for LP, most algorithms provide the optimal dual solution simultaneously with the primal solution. Primal and dual feasibility (or, alternatively, identical objective values) prove optimality. For IP the analogous method would be to use the optimal Chvátal function, as discussed in Sect. 2.2. But this would be, at least, exponentially difficult to calculate.

As mentioned above there are truly 'intractable' problems, i.e. those which are *unsolvable* as discussed in Chapter 1, e.g. undecidable problem classes such as establishing the truth or otherwise of all statements in formalised arithmetic. However, some 'less ambitious' problem classes are also unsolvable, e.g. general IP with quadratic constraints and objective.

A number of other observations about the complexity of the problems discussed in this book are in order. It has already been remarked in Sect. 2 that 'easy' problems (e.g. LP and matching) appear to have a low Chvátal rank (0 or 1) and 'difficult' problems (IP, set covering, TSP, satisfiability) unbounded Chvátal rank. Also polynomially bounded classes usually require only a low-degree polynomial algorithm. For LP Karmarkar's algorithm is of degree 7 in the data m, n, A. Matching is of degree 3 in n (the number of nodes). This may be simply a reflection of the comprehensibility of the known algorithms rather than their potential existence. But it does suggest the possibility of high-order polynomially bounded algorithms for the 'difficult' (thought to be $\mathbb{NP}$ complete) problems. If this were the case the ediface would collapse and we have $\mathbb{P} = \mathbb{NP}$.

2.5 References and Further Work

The original text on LP is Dantzig [30]. It is still a major valuable text. Chvátal [24] and Martin [79] are also excellent texts together with many more. The Fourier–Motzkin method for LP is due to Fourier [38] and is discussed in Williams [114] and Martin [79]. The original statement of the simplex 'algorithm' cannot be guaranteed to terminate. The smallest possible example of a non-terminating instance

is given by Hall and McKinnon [51]. There are a number of ways of modifying the simplex algorithm to guarantee termination. The simplest such method is due to Bland [15]. Even when termination is guaranteed the simplex algorithm may take an exponential number of steps as is shown by examples due to Klee and Minty [69] and Jeroslow [61]. It is still possible that the simplex algorithm could be polynomially bounded by a new choice of 'pivoting rule'. Average case analysis of the simplex algorithm has been done by Smale [102]. The largest LP reported (arising in financial planning) and solved has 353 million constraints and 1010 million variables and is due to Gondzio and Grothey [46].

Nemhauser and Wolsey [86] is the standard text on IP. The extension of Fourier–Motzkin elimination to IP is described by Williams [111]. Schrijver [98] is a major theoretical text on LP and IP. Williams [110, 113, 119] gives formulations of many types of LP and IP models. Appa et al. [3] edit a survey of applications of IP ranging from vehicle routing, the design of telecom networks, supply chain design and operation, pharmaceutical manufacture, oilfield infrastructure design and planning, radiation treatment for cancer, multitarget tracking and molecular biology. The application of the branch-and-bound algorithm to IP is due to Land and Doig [71] and Dakin [29]. The formalisation and naming of the concepts of relaxation and fathoming is due to Geoffrion and Marsten [43]. It is possible to ensure that the branch-and-bound algorithm terminates by placing bounds on the values of the integer variables derived from vertex or extreme ray solutions of the LP relaxation (so long as the data are rational).

The concept of a Chvátal function and the fact that it closes the duality gap for a pure IP is due to Chvátal [23].

Gomory [45] gives the first method of solving PIP models by adding cuts resulting from solutions by the simplex algorithm. Marchand and Wolsey [76] discuss the generation of cuts for MIP models (which we do not discuss here). Tind and Wolsey [104] give a survey of IP duality. Balas [8] also discusses it. Williams [118] surveys the concept of duality for mathematics in general.

The set covering, packing and partitioning problems have been extensively studied by, for example, Balas and Ng [10], Cornuejols and Sassano [27] and Balas and Padberg [11]. Edmonds and Johnson [35] describe the matching problem, its extensions and complexity. It is also discussed by Pulleyblank [92]. Ryan [97] applies set-partitioning models to very large aircrew scheduling problems. The practical solving of non-linear models by IP is discussed by, among others, Beale and Tomlin [13]. The knapsack problem is surveyed in Martello and Toth [77]. Facets of the 0–1 knapsack problem are given by Balas [6], Wolsey [125] and Hammer et al. [52]. The TSP has been extensively studied. A standard edited volume is Lawler et al. [74]. Orman and Williams [87] give and compare eight different formulations of the TSP as IPs. Applegate et al. [4] give a comprehensive computational study of the TSP.

Computational complexity is discussed in Aho et al. [1].

The concept of $\mathbb{NP}$ completeness is due to Cook [26] and Karp [67]. Garey and Johnson [41] is the standard text on the subject. Polynomial algorithms for LP are given by Khachian [68] and Karmarkar [66]. Depending on the exact implementation, Khachian's ellipsoid algorithm is bounded by a polynomial

of degree $(n^8 \log_2 \max(A_{ij}, c_j, b_i))$ and Karmarkar's algorithm is of degree $(n^7 \log_2 \max(A_{ij}, c_j, b_i))$. Jeroslow [60] shows that IP with quadratic constraints is unsolvable. Fourier–Motzkin elimination is of complexity $m^{(2^n)}$. The aggregation of pure IP models with equality constraints into knapsack models is due to Bradley [20]. A popular discussion of the creation of Gödel numbers is contained in Nagel and Newman [85].

2.6 Exercises

2.6.1 An LP can be regarded as an inference problem of finding the strongest inequality on the objective which is implied by the constraints. Express this problem in the predicate calculus applying it to Example 2.1.

2.6.2 Convert the predicate calculus representation in Exercise 2.6.1 to that of Example 2.3 using the rules of the predicate calculus.

2.6.3 Convert the dual model (2.23)–(2.27) to a *maximisation* subject to '$\leqslant$' constraints. Create the dual of this model and show it is the same as that in Example 2.1.

2.6.4 Represent the dual model (2.23)–(2.27) geometrically.

2.6.5 Apply the method of Example 2.3 to 'solve' Example 2.5 with objective (2.36).

2.6.6 Create the dual of Example 2.6 and represent it geometrically to show that is unbounded.

2.6.7 Show that
Maximise

$$x_1 + x_2$$

subject to

$$x_1 - x_2 \leqslant -1$$
$$-x_1 + x_2 \leqslant -1$$
$$x_1, x_2 \geqslant 0$$

$$x_1, x_2 \epsilon\ \mathbb{R}$$

and its dual are both infeasible.

2.6.8 Develop nodes 2 and 7 of Example 2.8 and show that they do not produce integer solutions.

2.6.9 Solve Example 2.8 by branch-and-bound, branching on x_2 at the top node.

2.6.10 Show that the following class of IP models will take an exponential (as a function of n) number of nodes to 'solve'

Maximise

$$x_1 + x_2 + \cdots + x_n$$

subject to

$$x_1, x_2, \cdots, x_n \geqslant 0$$
$$2x_1 + 2x_2 + \cdots + 2x_n = 3$$
$$x_1, x_2, \cdots, x_n \geqslant 0$$
$$x_1, x_2, \ldots, x_n \epsilon \mathbb{Z}$$

2.6.11 Use the method illustrated in Example 2.7 to demonstrate that Example 2.9 has no feasible (integer) solution.

2.6.12 Show by branch-and-bound that statement (1.83) is false.

2.6.13 Answer the following questions. If an answer is negative give a counter example.

i. If an IP is solvable (not infeasible or unbounded) is the corresponding LP relaxation solvable?
ii. If an IP is infeasible is the corresponding LP relaxation infeasible?
iii. If an IP is unbounded is the corresponding LP relaxation unbounded?
iv. If an LP relaxation of an IP is infeasible is the IP infeasible?
v. If an LP relaxation of an IP is unbounded is the IP unbounded?

2.6.14 Derive the optimal objective bound (i.e. optimal objective value) for the IP Example 2.7 as a combination of rounding and linear combinations of the original constraints. Give the Chvátal rank for the model.

2.6.15 By means of the class of models

Maximise

$$y$$

subject to

$$y - ax \leqslant 0$$
$$y + ax \leqslant a$$
$$y \geqslant 0, x, y \epsilon \mathbb{Z}$$

show that the class of two-variable pure IP models is of unbounded rank.

2.6.16 If a pure IP model with all equality constraints can be aggregated into an equality-constrained knapsack model why is the same not necessarily true for pure IP models with inequality constraints and a knapsack model with an inequality constraint?

2.6.17 Solve the model from Example 2.16 by branch-and-bound.

2.6.18 Show, using the decision procedure described in Example 1.8, that the LP relaxation of Example 1.9 **is** true.

2.6.19 Represent Example 1.8 geometrically.

Chapter 3
Modelling in Logic for Integer Programming

3.1 Logic Connectives and IP Constraints

We can regard an LP, or IP, constraint $\Sigma_j a_{ij} x_j (\leqslant = \geqslant) b_i$ as an atomic proposition. LP is concerned with optimising over a *conjunction* of such constraints. If, however, we wish to allow a more general logical statement about such constraints we require an IP model.

In order to systematise the modelling of such conditions it is convenient to introduce 0–1 integer variables to represent the satisfaction, or otherwise, of such constraints. When this is done it is usually necessary to know the upper or lower bounds for the left-hand side of the constraint above, i.e.

$$m_i \leqslant \sum_j a_{ij} x_j \leqslant M_i \tag{3.1}$$

Such bounds can usually be derived from knowledge of the problem or known bounds on the variables. It is important for the bounds not to place unwarranted restrictions, but computationally desirable for them to be as 'tight' as possible. In fact the derivation of as small 'big M' (or large 'little m') values as possible is of central importance, as this affects the strength of the LP relaxation, and consequently the amount of computation in the subsequent IP optimisation by, for example, branch-and-bound. This aspect is discussed in Sect. 3.3. If it is desired to get the tightest values of M and m possible then one could, of course, resort to the, possibly computationally costly, approach of maximising and minimising $\sum_j a_{ij} x_j$ subject to the LP relaxations of the other constraints in the problem to obtain M and m, respectively. In the 'convex' method of formulation, described in Sect. 3.4, it is possible to dispense with M and m altogether resulting in the LP relaxation yielding integer solutions.

If it is impossible to place either one or both bounds on the quantity then the condition may be *MIP unrepresentable* as discussed in Sect. 3.2. In order to describe the necessary modelling we use the following example.

Example 3.1 Represent the following conditions by (mixed) IP constraints involving the variables in the constraints and the 0–1 variables δ_i:

H.P. Williams, *Logic and Integer Programming,* International Series in Operations Research & Management Science 130, DOI 10.1007/978-0-387-92280-5_3,
© Springer Science+Business Media, LLC 2009

i.

$$\sum_j a_{ij}x_j \leqslant b_i \rightarrow \delta_i = 1 \tag{3.2}$$

ii.

$$\delta_i = 1 \longrightarrow \sum_j a_{ij}x_j \leqslant b_i \tag{3.3}$$

iii.

$$\sum_j a_{ij}x_j \leqslant b_i \longleftrightarrow \delta_i = 1 \tag{3.4}$$

For illustration we are only considering the inequality '$\leqslant$'. It is straightforward to extend to '$\geqslant$' and '$=$' by converting these (in) equalities to '$\leqslant$'.

A convenient way of viewing condition (i) is by the *contrapositive* statement discussed in Sect. 1.3. The contrapositive of (3.2) is

$$\delta_i = 0 \rightarrow \tilde{} \left(\sum_j a_{ij}x_j \leqslant b_i \right) \tag{3.5}$$

which, in turn, can be written as

$$\delta_i = 0 \rightarrow \sum_j a_{ij}x_j > b_i \tag{3.6}$$

We do not normally use the strict inequalities '$<$' and '$>$' in LP and IP. This is because we wish to find values for variables which give *maxima* and *minima*, which may lie on a boundary. But for strict inequalities we may only be able to give *suprema* (least upper bound) or *infima* (greatest lower bound). However, we can write, to a degree of approximation, (3.6) (using '$\leqslant$' for convenience) as

$$\delta_i = 0 \rightarrow -\sum_j a_{ij}x_j \leqslant -b_i - \varepsilon \tag{3.7}$$

where ε is a suitable small number. If the left-hand side of the inequality is all integer ε may be taken as 1 (and there be no approximation).

We wish to impose the condition that if $\delta_i = 0$ then $-\sum_j a_{ij}x_j + b_i + \varepsilon \leqslant 0$. The largest that the left-hand side of this expression can ever be is $-m + b + \varepsilon$. Hence we may write

$$-\sum_j a_{ij}x_j + b_i + \varepsilon \leqslant (-m_i + b_i + \varepsilon)\delta_i \tag{3.8}$$

Equation (3.8) is more conveniently written as

$$\sum_j a_{ij}x_j - (m_i - b_i - \varepsilon)\delta_i \geqslant b_i + \varepsilon \tag{3.9}$$

Therefore condition (i) above has been written as an IP constraint.

Equation (3.3) can be written, using analogous reasoning to that for (i), as

$$\sum_j a_{ij}x_j + (M_i - b_i)\delta_i \leqslant M_i \tag{3.10}$$

This gives condition (ii) as an IP constraint.

Finally condition (iii) is modelled by taking both constraints (3.9) and (3.10) together.

If any of (3.2), (3.3) or (3.4) involve a conjunction of the original constraints then the single 0–1 variable δ_i can be used to represent **all** of these constraints.

It is sometimes unnecessary to model the full equivalence condition (iii) between the satisfaction of a constraint and the value of a 0–1 variable since one of the implications may be guaranteed by optimality. For example this happens with the facility location problem discussed in Sect. 2.3. A facility (e.g. warehouse) will not be opened if it is not used since this would be a non-optimal thing to do. Hence the condition $\sum_j x_{ij} = 0 \rightarrow \delta_i = 0$ need not be modelled explicitly.

We are now in a position to model all the connectives of the propositional calculus, applied to LP or IP constraints, by first representing the constraints by 0–1 '*indicator*' variables δ_1 and δ_2 and modelling one of the conditions above. In many situations we may be able to suffice with modelling only condition (i) or condition (ii), but this will depend on the particular problem.

Because of the results in Sect. 1.3, we can suffice with modelling only a complete set of connectives. However, for convenience, we will model all the most commonly used connectives (Table 3.1).

Table 3.1 Connectives as 0–1 Constraints

$\vee$	$\cdot$	$-$	$\longrightarrow$	$\longleftrightarrow$
$\delta_1 + \delta_2 \geqslant 1$	$\delta_1 + \delta_2 = 2$	$\delta_1 = 0$	$\delta_1 \leqslant \delta_2$	$\delta_1 = \delta_2$

For the '$\vee$' connective if we have a disjunction of just two constraints then we could suffice with one 0–1 variable δ setting it to 0 to imply (or be implied by) one constraint and 1 for the other constraint. But we have used two 0–1 variables to demonstrate the generalisation to a disjunction of any number of constraints.

However if, in general, we have the disjunction of r constraints (or disjunction of r conjunctions of constraints) we could suffice with $\lceil \log_2 r \rceil$ (where '$\lceil\rceil$' represents the integer round-up operation), 0–1 variables since there will then be at least r settings of these variables. This is the subject of Exercise 3.7.1.

Example 3.2 shows how we may use these modelling devices.

Example 3.2 Model the following condition between LP constraints:

$$\left[\begin{pmatrix} 2x_1 + x_2 \leqslant 6 \\ x_2 \leqslant 4 \end{pmatrix} \vee \begin{pmatrix} x_1 + 3x_2 \leqslant 9 \\ 7x_1 + x_2 \leqslant 20 \end{pmatrix}\right] \cdot 2x_1 + 2x_2 \leqslant 9 \tag{3.11}$$

$$x_1, x_2 \geqslant 0 \tag{3.12}$$

$$x_1, x_2 \epsilon \mathbb{R}$$

It can be seen from the constraints above that $0 \leqslant x_1 \leqslant 3$, $0 \leqslant x_2 \leqslant 4$. Hence (numbering the five constraints in (3.11) as 1, 2, 3, 4, 5) we can give lower and upper bounds on their left-hand sides of $m_1 = m_2 = m_3 = m_4 = m_5 = 0$, $M_1 = 10$, $M_2 = 4$, $M_3 = 15$, $M_4 = 25$, $M_5 = 14$. It can be verified that these bounds apply. Further analysis would allow us to tighten these bounds to computationally beneficial effect. However we use these, easily obtained, bounds for illustration. This allows us to link 0–1 variables δ_i to the constraints. We only need model condition (ii) for each of them since we only need to make an implication in one direction, for each constraint. This gives

$$2x_1 + x_2 + 4\delta_1 \leqslant 10 \tag{3.13}$$

$$x_2 \leqslant 4 \tag{3.14}$$

$$x_1 + 3x_2 + 6\delta_2 \leqslant 15 \tag{3.15}$$

$$7x_1 + x_2 + 5\delta_2 \leqslant 25 \tag{3.16}$$

$$2x_1 + 2x_2 + 5\delta_3 \leqslant 14 \tag{3.17}$$

(Note that it is unnecessary to use variable δ_1 to imply the second constraint since it applies overall.)

It can be verified that setting a particular δ_i to 1 forces the corresponding constraints to hold. The logical relations between the constraints in (3.11) can then be written as

$$[(\delta_1 = 1) \vee (\delta_2 = 1)] \cdot \delta_3 = 1 \tag{3.18}$$

The logical condition can now be modelled by

$$\delta_1 + \delta_2 \geqslant 1 \tag{3.19}$$

$$\delta_3 = 1$$

If we have an objective then we have a MIP model. We solve this below.

Alternative and better ways of formulating such logical problems (including this example) are described in Sect. 3.3.

3.2 Disjunctive Programming

The logical connective that distinguishes IP from LP is '$\vee$'. It has been seen that '~' simply transforms inequalities to other (strict) inequalities that are usually modelled by approximating a non-strict inequality. The '$\cdot$' connective is implicitly present in all LP models. If one negates an '$=$' constraint (possibly first writing it as a conjunction of a '$\leqslant$' and a '$\geqslant$' constraint) one also obtains a disjunction. A *disjunction* is what requires the use of IP rather than LP. Hence the phrase '*disjunctive programming*' is sometimes used to characterise this type of modelling.

Although we will be showing, in this chapter, how to model disjunctive programmes as IPs it would be possible to solve them (inefficiently) using the method described (for the theory of dense linear order) in Example 2.3. This is the subject of Exercises 3.7.9 and 3.7.10.

3.2.1 A Geometrical Representation

It is helpful to illustrate such models geometrically. We do this by representing Example 3.2 geometrically, in Fig. 3.1.

The boundaries of all the inequalities in (3.11) are marked and the feasible region, defined by the logical statement of inequalities, is OABCDEFGH. Note that this is really a Venn diagram (discussed in Sect. 1.3) as a result of the conjunctions and disjunctions. It is *non-convex,* unlike LP feasible regions that are always convex. A convex region is one in which any line between two points in the region always, itself, lies in the region. In Figs. 3.2 and 3.3 we illustrate the distinction.

Non-convex regions arise in a number of contexts. In non-linear programming the feasible regions may be non-convex and IP used, possibly approximating the feasible region by piecewise linear segments. IP has the advantage that it finds global, as opposed to local, optima as discussed in Sect. 2.3. If a region is non-convex, and approximated to in this way, then the convex region can be partitioned into the union of convex sets. Each convex set can be represented as the *conjunction* of LP constraints. Hence the overall region is a *disjunction* of these conjunctions giving us the constraints of a disjunctive programme. Therefore there is a close connection between IP and (non-convex) non-linear programming.

There is also a distinction to be made between convex and non-convex objectives. If the region above each contour, representing equal values of the objective function, is a convex set then the objective function is said to be convex. Minimising a convex function over a convex region (or equivalently maximising a concave function over a convex region) guarantees a global optimum from the LP relaxation. We therefore

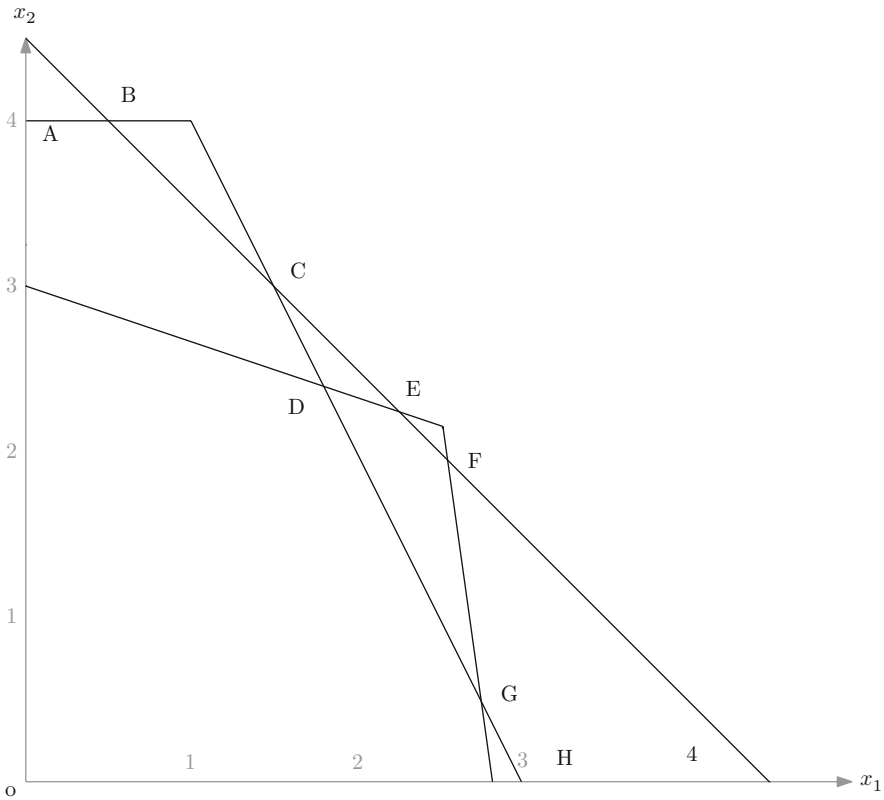


Fig. 3.1 The feasible region of a disjunctive programme

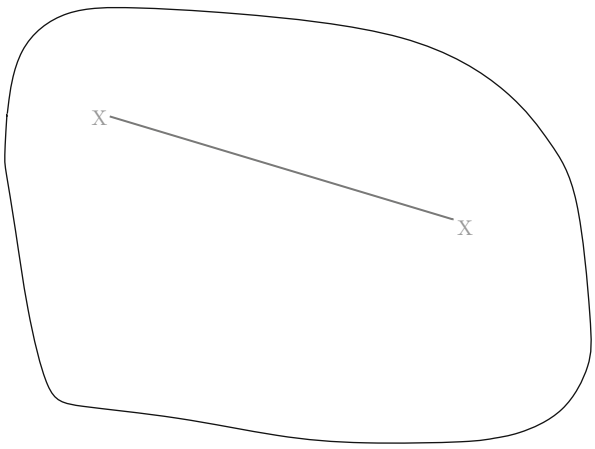


Fig. 3.2 A convex region

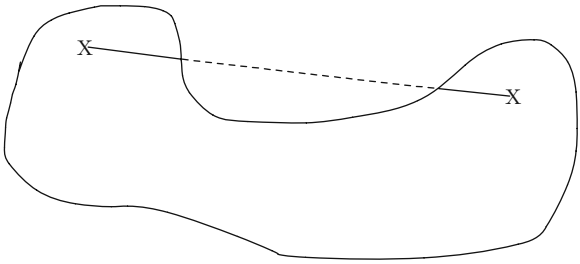


Fig. 3.3 A non-convex region

do not need to stipulate, and model condition (2.115) given in Chapter 2, or use integer variables. Such non-linear convex models arise in a number of contexts, e.g. when we have decreasing returns to scale.

In order to illustrate disjunctive programming further we solve Example 3.2 with an objective.

Example 3.3 Given the constraints in Example 3.2

Maximise

$$3x_1 + 2x_2 \tag{3.20}$$

For convenience we restate the MIP constraints we created

$$2x_1 + x_2 + 4\delta_1 \leqslant 10 \tag{3.21}$$
$$x_2 \leqslant 4 \tag{3.22}$$
$$x_1 + 3x_2 + 6\delta_2 \leqslant 15 \tag{3.23}$$
$$7x_1 + x_2 + 5\delta_2 \leqslant 25 \tag{3.24}$$
$$2x_1 + 2x_2 + 5\delta_3 \leqslant 14 \tag{3.25}$$
$$\delta_1 + \delta_2 \geqslant 1 \tag{3.26}$$
$$\delta_3 = 1 \tag{3.27}$$

Solving the LP relaxation of this model gives

$$x_1 = 3.086, \quad x_2 = 1.414, \quad \delta_1 = 0.603, \quad \delta_2 = 0.397, \quad \delta_3 = 1,$$
$$\text{Objective} = 12.086 \tag{3.28}$$

Proceeding (e.g. by branch-and-bound) to the integer optimum gives

$$x_1 = 2\frac{7}{12}, \quad x_2 = 1\frac{11}{12}, \quad \delta_1 = 0, \quad \delta_2 = 1, \quad \delta_3 = 1,$$
$$\text{Objective} = 11\frac{7}{12} \tag{3.29}$$

It is instructive to view these solutions geometrically (in the space of x_1 and x_2) as is done in Fig. 3.4.

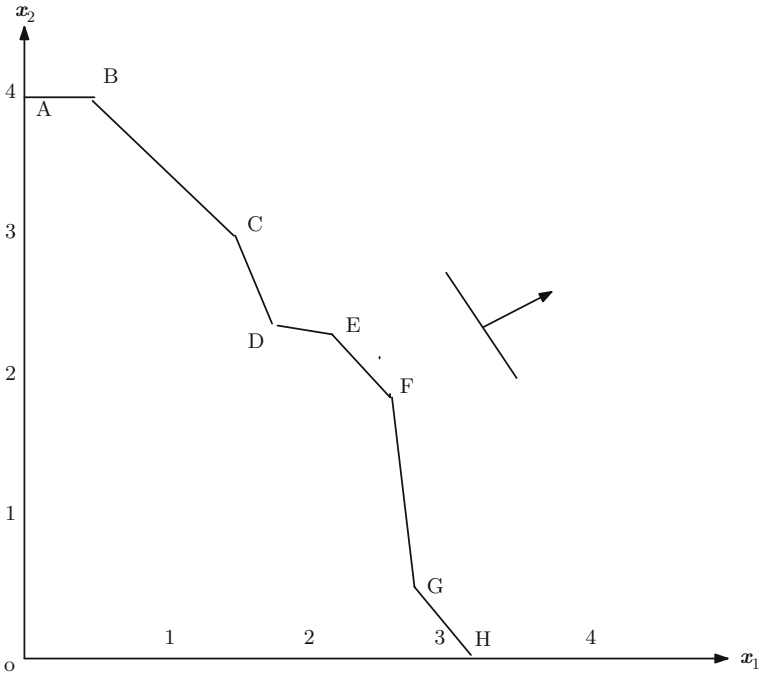


Fig. 3.4 A disjunctive programme

The optimal integer solution corresponds to vertex F in Fig. 3.4. Note that if this had represented a non-linear problem vertex H would have represented a local optimum (with a lower objective value). Traditional non-linear programming algorithms could well produce this solution. If we use branch-and-bound on the IP formulation then vertex H will correspond to another integer solution which might well be produced in the tree search before that at vertex F is found.

3.2.2 Mixed IP Representability

When is a problem representable as a MIP? We are going to restrict our attention to the possibility of modelling using *bounded* integer variables. In Chapter 2 it was shown that bounded integer variables can be expressed using only 0–1 integer variables. Hence we consider when it is possible to model, using continuous and 0–1 integer variables only. Such problems are sometimes referred to as bounded MIP representable but we will simply refer to these problems as MIP representable. We have seen that, in some circumstances, e.g. Example 3.2, it is possible to model a logical statement involving LP constraints as a conjunction of MIP constraints. But

we relied on some of the numerical bounds given in (3.1) in order to do this. If we know such bounds then it is possible to model the conditions in this way (possibly approximating strict inequalities if necessary). In some circumstances we do not know such bounds. Also the objective function may make the problem non-MIP representable. It may still be possible to model as a MIP but only if certain conditions apply, which we discuss here. Before doing this we give a simple logical condition which is not MIP representable.

Example 3.4 A non-MIP-representable condition

$$x = 0 \vee y = 0 \tag{3.30}$$

$$x, y \in \mathbb{R}$$

Since we do not have bounds on the values of x and y we cannot, obviously, model this disjunction by introducing 0–1 variables. Even if we place single bounds on each variable (e.g. lower bounds of 0) the condition is still not representable.

Example 3.5 A MIP-representable disjunctive programme with an open feasible region

Minimise

$$x + 3y \tag{3.31}$$

subject to

$$\begin{pmatrix} 2x - y \geqslant -3 \\ -x + y \geqslant 2 \\ x \geqslant 0 \end{pmatrix} \vee \begin{pmatrix} 4x - 2y \geqslant 4 \\ -2x + 2y \geqslant -2 \\ x + y \geqslant 2 \end{pmatrix} \vee \begin{pmatrix} 6x - 3y \geqslant 0 \\ -3x + 3y \geqslant 1 \\ 2x + 3y \geqslant 3 \end{pmatrix} \tag{3.32}$$

$$x, y \in \mathbb{R}$$

The feasible region is illustrated in Fig. 3.5. It can be seen that each clause in (3.32) gives rise to an open feasible region, as therefore does their union. However, Example 3.5 is MIP representable, unlike Example 3.4.

A MIP model representing Example 3.5 is

Minimise

$$x + 3y \tag{3.33}$$

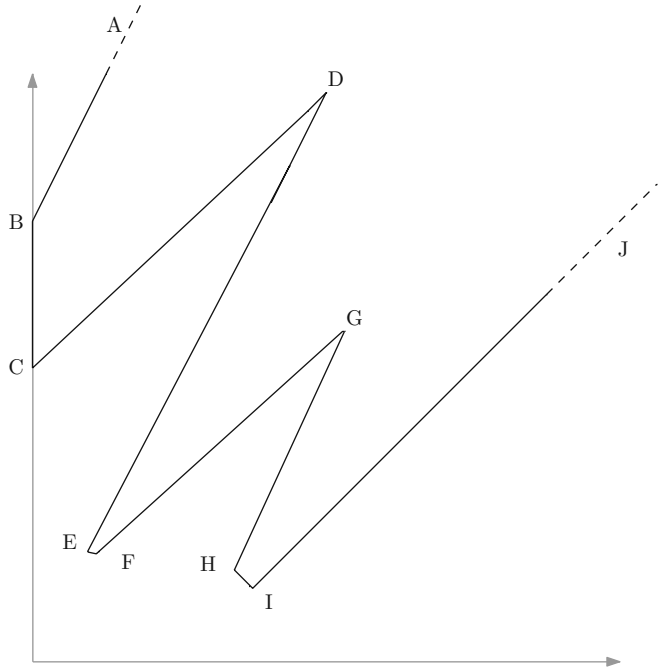


Fig. 3.5 Disjunctive constraints with an open feasible region

subject to

$$-x + y \geqslant -1 + 3\delta_1 \tag{3.34}$$
$$x \geqslant 0 \tag{3.35}$$
$$4x - 2y \geqslant -6 + 10\delta_2 \tag{3.36}$$
$$8x + 8y \geqslant 9 + 7\delta_2 \tag{3.37}$$
$$6x - 3y \geqslant -9 + 9\delta_3 \tag{3.38}$$
$$-3x + 3y \geqslant -3 + 4\delta_3 \tag{3.39}$$
$$2x + 3y \geqslant 3 \tag{3.40}$$
$$\delta_1 + \delta_2 + \delta_3 \geqslant 1 \tag{3.41}$$

$$\delta_1, \delta_2, \delta_3 \in \{0, 1\}$$

Constraint (3.41) forces at least one of $\delta_1, \delta_2, \delta_3$ to take the value 1. It can be verified that $\delta_1 = 1$ forces the constraints in the first clause of (3.32) to hold (but does not exclude the corresponding constraints holding in the other clauses). Similarly $\delta_2 = 1$ and $\delta_3 = 1$, respectively, force the second and third clauses to hold.

Why is Example 3.5 MIP representable while Example 3.4 is not? The reason is that the feasible region of Example 3.5 is the union of polyhedra with the same *recession directions.* A recession direction of a polyhedron (in this two-dimensional example) is a direction (p, q) such that for any point (a, b) in the polyhedron all points $(a+\lambda p, b+\lambda q)$ for all $\lambda \geqslant 0$ are also in the polyhedron. Recession directions are drawn for a polyhedron in Fig. 3.6.

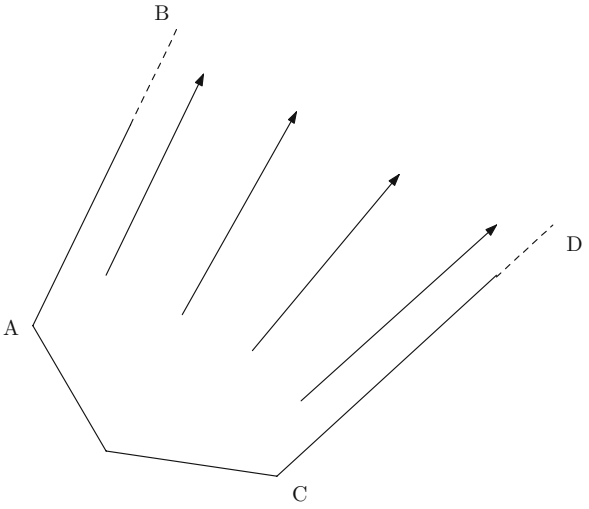


Fig. 3.6 Recession directions of an open polyhedron

Possible recession directions are those between the extreme rays AB and CD. In Fig. 3.5 it can be seen that the three polyhedra, whose union makes up the feasible region, have exactly the same recession directions since they all have extreme rays pointing in the same directions. We show below why this property makes the corresponding optimisation problems MIP representable. Before doing this, however, observe that Example 3.4 can be regarded as the union of two polyhedra $x = 0$ and $y = 0$ each of which have two, *different*, recession directions. Hence this example is not MIP representable.

If all the polyhedra, whose union makes up the feasible region of a problem, are closed then each of the polyhedra has no recession directions and the conditions are satisfied vacuously making the problem MIP representable. This is the case with Example 3.3.

Although all the above discussion has been illustrated by two-dimensional examples the concepts and results extend to any number of dimensions.

For completeness we solve the model resulting from Example 3.5.

The LP relaxation solution is

$$\begin{aligned} &x = 0.719, \quad y = 0.521, \quad \delta_1 = 0.267, \quad \delta_2 = 0131, \quad \delta_3 = 0.602, \\ &\text{Objective} = 2.281 \end{aligned} \tag{3.42}$$

The integer optimum is

$$x = \frac{2}{5}, \quad y = \frac{11}{15}, \quad \delta_1 = 0, \quad \delta_2 = 0, \quad \delta_3 = 1, \quad \text{Objective} = 2\frac{3}{5} \tag{3.43}$$

The integer solution (3.43) corresponds to the point F in Fig. 3.5. Clearly the third clause in the disjunction (3.32) is true.

We now show that the same recession directions for a set of polyhedra guarantees their disjunction is MIP representable.

If a set of polyhedra all have the same recession directions then an open facet of any of the polyhedra must have extreme rays which are contained in each of the other polyhedra. Consider an open facet from one of the polyhedra defined by the constraint

$$\sum_j a_{1j} x_j \geqslant b_1 \tag{3.44}$$

Suppose we were to

Minimise

$$\sum_j a_{1j} x_j \tag{3.45}$$

subject to the constraints defining each of the other polyhedra in turn. If any of the resulting LPs were unbounded, then the directions along which (3.45) reduces indefinitely would define extreme rays of the other polyhedron. But such extreme rays must also be a recession directions of (3.45) contradicting the unboundedness. Hence we can determine finite values of m_i.

The optimal value of (3.45) is determined by its value at the vertex at the origin of this extreme ray (so long as the facet is 'pointed'. If not we can choose a value at any point in the polyhedron). Hence we can obtain a (finite) lower bound, m_1, on the value of $\sum_j, a_{1j} x_j$. Similarly we can find finite lower bounds m_2 , ..., m_n, on the values of the left-hand sides of the other 'open' constraints.

Note that if the number of distinct recession directions, on an open facet, is at least as great as the dimension of that facet then the corresponding facets must all be parallel and the m_i can be taken as $\min_{k \neq i}(b_k)$ (Exercises 3.7.2 and 3.7.3). This is the case for Example 3.5.

We can force all of the constraints in each conjunctive clause to hold by setting the corresponding 0–1 variable $\delta_i = 1$ in MIP constraints

$$\sum_j a_{ij} x_j - (b_i - m_i)\delta_i \geqslant m_i \quad \forall i \tag{3.46}$$

together with

$$\sum_i \delta_i \geqslant 1 \tag{3.47}$$

i.e. the same 0–1 variable is used for each constraint (open and closed) in a conjunctive clause.

Note that we can, in effect, confine ourselves to the same m_i (or M_i for '$\leqslant$' constraints) which we would have used in the 'closed parts' of the corresponding polytopes.

For constraints that do not contain extreme rays of the polyhedra we can obtain bounds on the values of the variables in them and formulate the corresponding MIP constraints as described in Sect. 3.1.

Conversely, if we have a MIP representation of a model, then a typical constraint will be

$$\sum_{j\in J} a_j x_j - \sum_{i\in I} b_i \delta_i \geqslant b \tag{3.48}$$

$$x_j \in \mathbb{R},\ \delta_i \in \{0, 1\} \quad \forall j \in J, i \in I$$

Setting the δ_i to different values gives a series of parallel constraints.

Some, or all, of these constraints may represent closed hyperplanes. Some may be redundant in the presence of other constraints. However, if any represent open facets, of some of the component polyhedra in the implied disjunction, they must all represent open facets. What's more they will be bounded by some extreme rays which must be parallel to the corresponding extreme rays in the other polyhedra in the implied disjunctions. This argument applies to all the constraints in (3.48). Hence the constraints in (3.48) represent a disjunction of polyhedra that are either all closed or, if not, have the same recession directions.

It remains to consider when the objective function, taken in conjunction with the constraints, still makes a problem MIP representable. If the objective is linear and expressed in the original variables (not involving the appended 0–1 variables) then there is no problem. This is the case with Examples 3.3 and 3.5. However, it may be the case that we wish to model a piecewise linear or a discontinuous objective. For example the non-linear expression given in Example 2.17 might have occurred in the objective function. It would then be replaced by a variable, with the necessary extra variables and constraints added to the model. The representability problem would then be the same as that for a logical system of constraints. If the objective is discontinuous then the situation may be more complex. This happens, for example, in the facility location problem discussed in Example 2.12. This problem is a special case of the *fixed charge problem.* We consider this problem in its simplest form in Example 3.6.

Example 3.6 Formulate the following fixed charge problem as a MIP.

If an activity is carried out at a level $x > 0$ then it incurs a fixed cost f, irrespective of the level. However, if $x = 0$ then there is no fixed cost (and no revenue). To be realistic we might give the activity x a (positive) revenue of p.

Since we normally wish to minimise total cost (explicitly, or implicitly by maximising profit) we minimise z such that

$$z \geqslant 0 \quad \text{if } x = 0 \tag{3.49}$$

$$z \geqslant f - px \quad \text{if } x > 0 \tag{3.50}$$

$$z, x \epsilon \mathbb{R}$$

Equations (3.49) and (3.50) define the polyhedra, in Fig. 3.7, consisting of the positive z-axis and the region above the line AC. The second polyhedron clearly has recession directions different to that of the first polyhedron (which has a single recession direction). Therefore this polyhedron, and therefore the unbounded fixed charge problem, is not MIP representable. However, if we place an upper (M) bound on the value of x then the problem is MIP representable as the second polyhedron has the same, single, recession direction as the first. The union of these two polyhedra is known as the 'epigraph' of the cost function. So long as the epigraph is MIP representable (and the feasible region is MIP representable) then the minimisation problem is MIP representable. This gives rise to the model

Maximise

$$px - f\delta \tag{3.51}$$

subject to

$$x \leqslant M\delta \tag{3.52}$$

$$x \geqslant 0 \tag{3.53}$$

$$x \epsilon \mathbb{R}, \quad \delta \epsilon \{0, 1\}$$

Note that for the facility location model, in Example 2.12, both the customer demands and the warehouse capacities place upper bounds on the x variables, making the problem MIP representable.

3.3 Alternative Representations and Tightness of Constraints

Alternative representations of statements in the propositional calculus were discussed, at length, in Chapter 1. These different representations lead to different IP formulations. Example 2.10 demonstrates this.

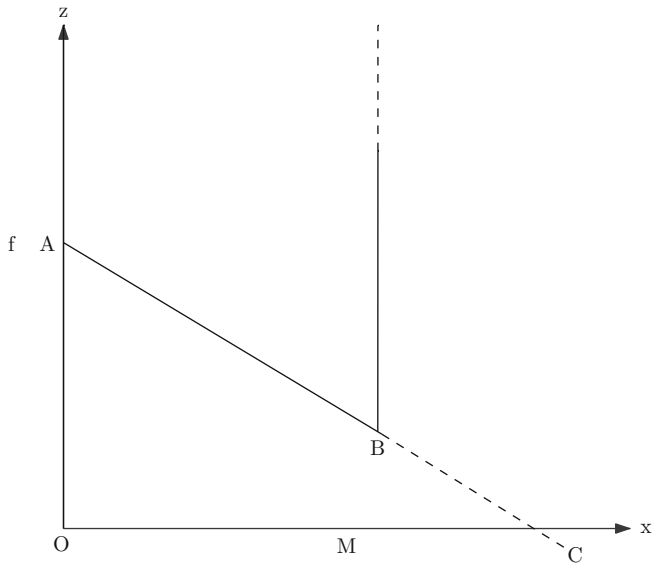


Fig. 3.7 Cost function for the fixed charge problem

Equation (2.80) models the logical condition

$$(\delta_1 = 1 \vee \delta_2 = 1) \longrightarrow \delta_3 = 0 \tag{3.54}$$

This statement is logically equivalent to

$$(\delta_1 = 1 \longrightarrow \delta_3 = 0) \cdot (\delta_2 = 1 \longrightarrow \delta_3 = 0) \tag{3.55}$$

(Exercise 3.7.4 is to verify this.)

However (3.55) leads, naturally, to the conjunction of constraints (2.82) and (2.83). Figures 2.12 and 2.13 demonstrate that this latter IP formulation represents the convex hull of feasible integer solutions, allowing one to use computationally easier LP.

We now investigate the respective merits of IP formulations resulting from DNF and CNF representations of logical conditions. Following Chapter 2 we refer to convex hull formulations as ‘sharp’ and formulations that are closer to the convex hull as ‘tighter’.

3.3.1 Disjunctive Versus Conjunctive Normal Form

Example 3.7 Formulate Example 3.2 in DNF and create, and solve, the corresponding IP model.

The logical representation is

$$\left[\begin{pmatrix} 2x_1 + x_2 \leqslant 6 \\ x_2 \leqslant 4 \\ 2x_1 + 2x_2 \leqslant 9 \end{pmatrix} \vee \begin{pmatrix} x_1 + 3x_2 \leqslant 9 \\ 7x_1 + x_2 \leqslant 20 \\ 2x_1 + 2x_2 \leqslant 9 \end{pmatrix}\right] \tag{3.56}$$

$$x_1, x_2 \geqslant 0 \tag{3.57}$$

$$x_1, x_2 \epsilon \mathbb{R}$$

Using objective (3.20) and representing the two clauses in the disjunction by 0–1 variables δ_1 and δ_2, respectively, give the model

Maximise

$$3x_1 + 2x_2 \tag{3.58}$$

subject to

$$2x_1 + x_2 + 4\delta_1 \leqslant 10 \tag{3.59}$$
$$x_2 \leqslant 4 \tag{3.60}$$
$$2x_1 + 2x_2 + 5\delta_1 \leqslant 14 \tag{3.61}$$
$$x_1 + 3x_2 + 6\delta_2 \leqslant 15 \tag{3.62}$$
$$7x_1 + x_2 + 5\delta_2 \leqslant 25 \tag{3.63}$$
$$2x_1 + 2x_2 + 5\delta_2 \leqslant 14 \tag{3.64}$$
$$\delta_1 + \delta_2 \geqslant 1 \tag{3.65}$$
$$x_1, x_2 \geqslant 0 \tag{3.66}$$

$$x_1, x_2 \epsilon \mathbb{R}$$
$$\delta_1, \delta_2 \epsilon \{0, 1\}$$

If we solve this alternative formulation we first obtain the LP relaxation solution

$$x_1 = 2.759, \quad x_2 = 2.795, \quad \delta_1 = 0.422, \quad \delta_2 = 0.578, \quad \text{Objective} = 13.867 \tag{3.67}$$

followed by the integer optimal solution given in (3.29).

The difference between the solution in (3.67) and (3.29) (the duality gap) is greater than that between (3.28) and (3.29). This indicates that this new formulation

is not as tight as the other formulation. In this sense it is not as good. In general branch-and-bound would take longer to obtain the integer optimum.

At the opposite extreme to the DNF given in Example 3.7 we consider CNF for this example.

Example 3.8 Express the logical statement in Example 3.7 (and 3.2) in CNF and formulate and solve the resultant IP model.

In CNF we have

$$\begin{gathered}
[2x_1 + x_2 \leqslant 6 \vee x_1 + 3x_2 \leqslant 9] \\
\cdot [2x_1 + x_2 \leqslant 6 \vee 7x_1 + x_2 \leqslant 20] \\
\cdot [x_2 \leqslant 4 \vee x_1 + 3x_2 \leqslant 9] \\
\cdot [x_2 \leqslant 4 \vee 7x_1 + x_2 \leqslant 20] \\
\cdot 2x_1 + 2x_2 \leqslant 9 \\
x_1, x_2 \geqslant 0 \\
x_1, x_2 \epsilon \mathbb{R}
\end{gathered} \tag{3.68}$$

Introducing 0–1 variables $\delta_{1_1}, \delta_{1_2}, \delta_{2_1}, \delta_{2_2}$ to distinguish between the constraints in the disjunctions (i.e. index i_j refers to the ith constraint in the jth clause of the DNF form of problem (3.56)) gives the model

Maximise

$$3x_1 + 2x_2 \tag{3.69}$$

subject to

$$2x_1 + x_2 + 4\delta_{1_1} \leqslant 10 \tag{3.70}$$
$$x_1 + 3x_2 + 6\delta_{1_2} \leqslant 15 \tag{3.71}$$
$$x_2 \leqslant 4 \tag{3.72}$$
$$7x_1 + x_2 + 5\delta_{2_2} \leqslant 25 \tag{3.73}$$
$$2x_1 + 2x_2 \leqslant 9 \tag{3.74}$$
$$\delta_{1_1} + \delta_{1_2} \geqslant 1 \tag{3.75}$$
$$\delta_{1_1} + \delta_{2_2} \geqslant 1 \tag{3.76}$$
$$\delta_{2_1} + \delta_{1_2} \geqslant 1 \tag{3.77}$$
$$\delta_{2_1} + \delta_{2_2} \geqslant 1 \tag{3.78}$$
$$x_1, x_2 \geqslant 0 \tag{3.79}$$

$$x_1, x_2 \epsilon \mathbb{R}$$
$$\delta_1, \delta_2 \epsilon \{0, 1\}$$

Solving the LP relaxation of this model (as with the DNF-based formulation) produces the solution (3.28), i.e. the same as for the DNF formulation showing them,

in this case, to be of equal strength. It can be shown that a DNF-based formulation (so long as the same M and m bounds are used for each constraint) will always be at least as tight, and sometimes tighter, than a CNF-based formulation. The reason for this is explained below. What's more the CNF formulation will often involve more 0–1 variables than the DNF formulation since we need r (or strictly $\lceil \log_2 r \rceil$) 0–1 variables to model each disjunction. If we have a conjunction of n such disjunctions then we will require a total of rn (or $\lceil \log_2 r \rceil n$) 0–1 variables, whereas in DNF a conjunction of m disjunctions could be modelled with m 0–1 variables. In practice substantial simplifications may be possible in either type of formulation, and the size of the CNF or DNF representation will depend on the problem, as discussed in Chapter 1. The tightness of the resultant IP is often of greater importance than the compactness of the model in terms of number of variables (and constraints). In Sect. 3.4 we show how it is possible to guarantee the production of a sharp IP formulation of a disjunctive programme, but often at the expense of a very large number of 0–1 variables.

In order to show why the LP relaxation associated with a DNF formulation is always at least as tight as that associated with a CNF formulation we consider general logical statements of constraints in both forms. For convenience we will consider all the constraints in the '$\leqslant$' form.

In DNF we have sets (indexed by j) of conjunctive clauses (indexed by i_j) each consisting of constraints $\sum_k a_{i_j k} x_{i_j k} \leqslant b_{i_j}$ for $i_j = 1, 2, ..., r_{i_j}$. These will be written (for economy of notation) as $f_{i_j} \leqslant 0$.

We introduce 0–1 variables δ_j. Each δ_j, when taking the value 1, forces **all** the constraints in the jth set to hold. The disjunction is formulated as

$$\sum_j \delta_j \geqslant 1 \tag{3.80}$$

The constraints are forced to hold by the MIP constraints

$$f_{i_j} + (M_{i_j} - b_{i_j})\,\delta_j \leqslant M_{i_j} - b_{i_j} \quad \forall j\ i_j \tag{3.81}$$

where the M_{i_j} are upper bounds on the values of the $f_{i_j} + b_{i_j}$ as derived in (3.10) or (in '$\leqslant$' form) (3.46).

In CNF, using the distributive rule (1.7), each original constraint will appear in each of the disjunctive clauses. Hence it is necessary to force each of them to hold by separate 0–1 variables δ_{i_j} taking the value 1 giving

$$f_{i_j} + (M_{i_j} - b_{i_j})\,\delta_{i_j} \leqslant M_{i_j} - b_{i_j} \quad \forall j\ i_j \tag{3.82}$$

together with the constraints (for each disjunctive clause)

$$\sum_{i_j} \delta_{i_j} \geqslant 1 \tag{3.83}$$

If we now impose the following extra constraints on this, CNF-based, formulation

$$\delta_{i_j} = \delta_j \quad \forall i_j \tag{3.84}$$

and so eliminate variables δ_{i_j} we obtain the DNF-based formulation. Hence a DNF-based formulation is always as tight as a CNF-based one and for some problem instances will be tighter.

When it is worth moving from CNF to DNF, in order to produce a tighter formulation, may depend on the particular problem. Even if a problem is not naturally stated in DNF or CNF it is usually better to move the conjunctive connective '$\cdot$' 'in' and the disjunctive connective '$\vee$' 'out' (using the distributive laws) in the logical representation so approximating to DNF.

Exercise 3.7.5 is to reformulate and solve Example 3.5 using CNF. It turns out that, for that example, we obtain the same LP relaxation solution (in the space of x and y) as that from the DNF formulation.

However for both Examples 3.2 and 3.5 we will show, in Sect. 3.4, a formulation whose LP relaxation always gives the optimal integer solution for all problem instances.

3.3.2 The Dual of a Disjunctive Programme

There is no, very satisfactory, dual of a general IP model. The most satisfactory dual of a PIP is the Chvátal dual discussed in Chapter 2. However, if a disjunctive programme is written in DNF then it has a fairly obvious dual. In order to illustrate this we define the dual of Example 3.7.

Example 3.9 Define the dual of Example 3.7.

We take the LP duals corresponding to each of the clauses in (3.56) taken with the objective (3.58) and take the solution giving the larger objective value. This gives

Minimise

$$\text{Max}(6y_1 + 4y_2 + 9y_3, 9y_4 + 20y_5 + 9y_6) \tag{3.85}$$

subject to

$$2y_1 + 2y_3 \geqslant 3 \tag{3.86}$$

$$y_1 + y_2 + 2y_3 \geqslant 2 \tag{3.87}$$

$$y_4 + 7y_5 + 2y_6 \geqslant 3 \tag{3.88}$$

$$3y_4 + y_5 + 2y_6 \geqslant 2 \tag{3.89}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \geqslant 0 \tag{3.90}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \in \mathbb{R}$$

Notice that, for this model to be feasible, the dual models corresponding to each of the clauses in the primal model must also be feasible. However, the original, primal, model could be regarded as feasible even if some of its clauses (up to all except one) were infeasible. As pointed out in Chapter 2 it is possible for an LP to have both its primal and dual infeasible. If this were the case for the LP resulting from one of the clauses we could have a feasible disjunctive programme with an infeasible dual. To get around this we could stipulate a 'regularity' condition for the duality theorem to apply, i.e. for no clause should both the primal and dual LPs be infeasible. Alternatively (and more satisfactorily) we could define the maximum and minimum objective values for an infeasible LP as $-\infty$ and ∞, respectively.

There is a useful notation for representing the above dual model. We can represent the operation $\text{Min}(a, b)$ as $a \oplus b$ (analogous to '$+$') and the operation $\text{Max}(a, b)$ as $a \oplus' b$. The above dual model can then be written as

Minimise

$$(6y_1 + 4y_2 + 9y_3) \oplus' (9y_4 + 20y_5 + 9y_6) \tag{3.91}$$

subject to

$$(2y_1 + 2y_3) \oplus (y_4 + 7y_5 + 2y_6) \geqslant 3 \tag{3.92}$$

$$(y_1 + y_2 + 2y_3) \oplus (3y_4 + y_5 + 2y_6) \geqslant 2 \tag{3.93}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \geqslant 0 \tag{3.94}$$

$$y_1, y_2, y_3, y_4, y_5, y_6 \in \mathbb{R}$$

This can be regarded as an extension of the dual of an LP. There as we 'pass over' a conjunction, looking down a column, the corresponding dual operation is '$+$'. Here we extend this so that when we pass over a disjunction the dual operation is '$\oplus$'. The need to use '$\oplus$' '(min)' and '$\oplus'$' '(max)' arises from the fact that the right-hand-side coefficients in an LP are conventionally written on the right (!). If they had been written, together with the variables, on the left then we could have sufficed with '$\oplus$'.

Obviously we have to amend our definition of the dual of a disjunctive programme to deal with the case of a minimisation (of the primal model) and '$\geqslant$' constraints. This is straightforward and not covered here.

This dual of a disjunctive programme obviously captures many of the features of the dual of an LP, stated in Sect. 2.1, i.e. equal objective values if both are solvable (subject to the proviso discussed) and symmetry (it can be shown that the dual of the dual is the primal: see Exercise 3.7.7).

An algebra, known as 'minimax' algebra has been developed around the use of symbols '$\oplus$' and '$\oplus'$'. References are given in Sect. 3.6.

3.4 Convexification of an IP Model

The concept of a convex hull can be extended beyond that of the smallest convex region containing feasible integer points. If we restrict ourselves to considering the disjunctions of constraints in the original space in which they are stated (i.e. x_1 and x_2 in the case of Example 3.2 and x and y in the case of Example 3.5) then we can define the *convex hull* of the disjunction of the resultant polyhedra. It is the smallest convex set containing all the polyhedra making up the disjunction. We draw the convex hulls corresponding to the disjunction of polyhedra in Figs. 3.4 and 3.5 in Figs. 3.8 and 3.9, respectively.

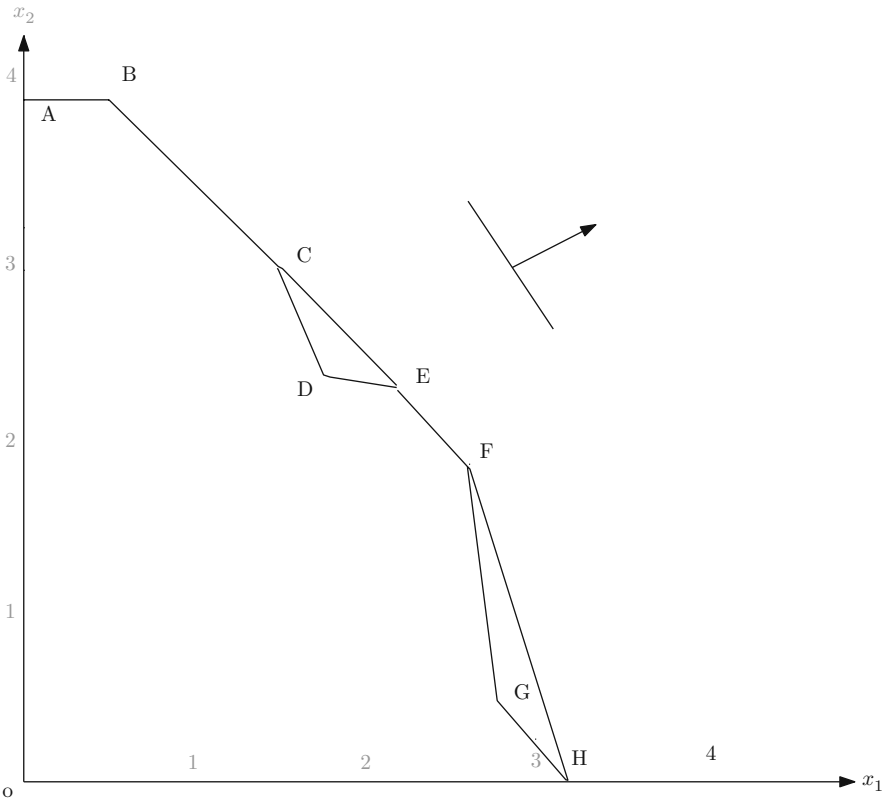


Fig. 3.8 The convex hull for Example 3.2

If we could append the extra constraints to represent the convex hulls then we could solve these examples as LPs producing the vertex solutions at F and F, respectively. In general (when there is no visualisable geometrical representation), the creation of constraints representing convex hulls is very difficult. We have given different IP formulations involving the 0–1 variables. These have different convex hulls (in these higher dimensional spaces). It is sometimes possible to project their

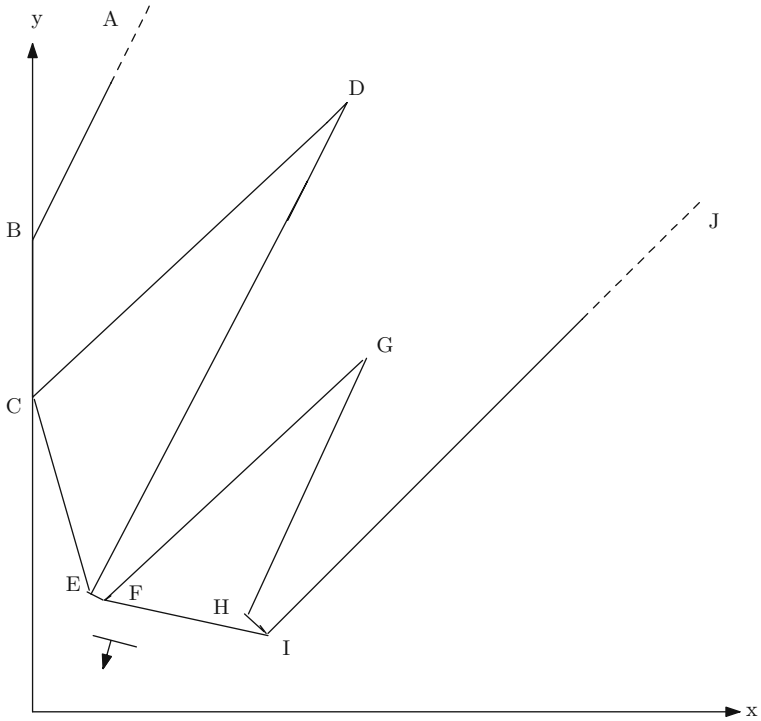


Fig. 3.9 The convex hull for Example 3.5

LP relaxations down (using the method described in Sect. 2.1) into the spaces of the original problems to give approximations to the convex hulls.

However, there is another, powerful, method of reformulating a disjunctive programme to give its convex hull representation by introducing (possibly a very large number of) new variables.

3.4.1 Splitting Variables

Example 2.10 demonstrates that 'disaggregating' constraints can improve the tightness of the LP relaxation. This can be taken further if we first split variables, in the original space of a disjunctive programme, into components. We demonstrate this by using Example 3.5 to produce what has become known as a 'disjunctive' formulation (although all the formulations involving '$\vee$' are, in a sense, disjunctive).

Example 3.10 Formulate Example 3.5 as a MIP by splitting the variables into components for each clause in the disjunction to produce a 'disjunctive' formulation.

Each of the variables x and y is split into a component for each of the three (conjunctive) clauses in (3.32) by the constraints

$$x_1 + x_2 + x_3 = x \tag{3.95}$$
$$y_1 + y_2 + y_3 = y \tag{3.96}$$

Three 0–1 variables are used to apply the clauses of the disjunction in the following way:

$$2x_1 - y_1 \geqslant -3\delta_1 \tag{3.97}$$
$$-x_1 + y_1 \geqslant 2\delta_1 \tag{3.98}$$
$$x_1 \geqslant 0 \tag{3.99}$$
$$4x_2 - 2y_2 \geqslant 4\delta_2 \tag{3.100}$$
$$-2x_2 + 2y_2 \geqslant -2\delta_2 \tag{3.101}$$
$$x_2 + y_2 \geqslant 2\delta_2 \tag{3.102}$$
$$6x_3 - 3y_3 \geqslant 0 \tag{3.103}$$
$$-3x_3 + 3y_3 \geqslant \delta_3 \tag{3.104}$$
$$2x_3 + 3y_3 \geqslant 3\delta_3 \tag{3.105}$$
$$\delta_1 + \delta_2 + \delta_3 = 1 \tag{3.106}$$

By virtue of (3.106) exactly one of $\delta_1, \delta_2, \delta_3$ will take the value 1, the others taking the value 0. Hence one of the clauses will hold for the corresponding component variables. For the other clauses the right-hand sides of the constraints (in the corresponding component variables) will become 0.

Suppose, for example, $\delta_1 = 0, \delta_2 = 1, \delta_3 = 0$ then the first of the extreme ray constraints in each clause are

$$2x_1 - y_1 \geqslant 0 \tag{3.107}$$
$$4x_2 - 2y_2 \geqslant 4 \tag{3.108}$$
$$6x_3 - 3y_3 \geqslant 0 \tag{3.109}$$

Adding these constraints together (in suitable multiples), and using (3.95) and (3.96) gives

$$4x - 2y \geqslant 4 \tag{3.110}$$

i.e. the first extreme ray constraint in the second clause, in the space of the original variables.

Similarly, adding together the second extreme ray constraints in each clause gives

$$-2x + 2y \geqslant -2 \tag{3.111}$$

i.e. the second extreme ray constraint in the second clause, in the space of the original variables.

Since the extreme ray constraints corresponding to $\delta_1 = 0$ and $\delta_3 = 0$ go through the origin they form a *cone* in the space of the appropriate component variables. The non-extreme ray constraints, in the components corresponding to these clauses, also go through the origin. They are

$$x_1 \geqslant 0 \tag{3.112}$$

$$x_2 + y_2 \geqslant 0 \tag{3.113}$$

$$2x_3 + 3y_3 \geqslant 0 \tag{3.114}$$

We illustrate the situation, for the third clause, in Fig. 3.10.

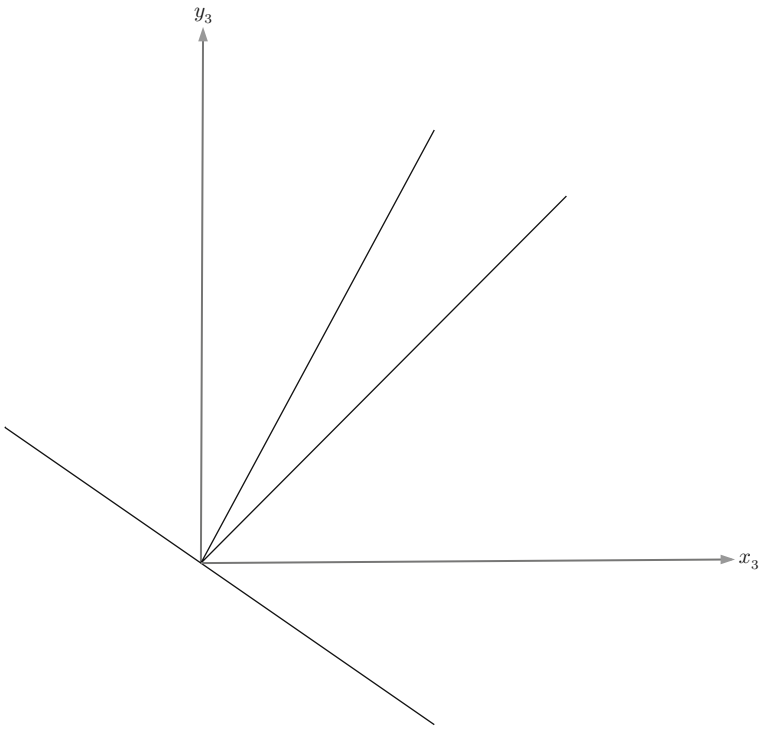


Fig. 3.10 Representation of third clause in split variables

This implies that the component variables in the clauses for which $\delta_i = 0$ are 0 when these constraints are satisfied as equalities. Therefore, as a result of (3.95) and (3.96), the appropriate constraints apply for the relevant clause in the space of the original variables. For this example we would have

$$x + y \geqslant 2 \tag{3.115}$$

The constraints of (3.34)–(3.41) are therefore relaxations of (3.97)–(3.106). Hence our new (disjunctive) formulation is at least as tight as the DNF formulation. Solving the LP relaxation of this new formulation gives the integer solution (3.43) showing it to be sharp.

If all the component polyhedra of a disjunction are closed then all except one of the polyhedra, in the component variables, reduce to the point at the origin, when the corresponding δ_i variables are 0. Therefore the clause corresponding to $\delta_i = 1$ applies in the space of the original (non-split) variables.

Although this type of formulation introduces many new variables (a component for each variable in each clause, whether or not the clause contained the variable in the first place), in practice many of them can often be shown to be redundant leading to a vast reduction in the size of the model. Such reductions depend on the individual type of model and are difficult to generalise.

A 'disjunctive' formulation is always sharp (it models the convex hull). We now show why this is the case. In order to do this we consider the dual of the LP relaxation of the model in Example 3.10.

Example 3.11 Create the dual of the LP relaxation of the model created in Example 3.10, together with objective (3.31) (written in the component variables).

Representing the dual variables on constraints (3.97)–(3.106) as $u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3, z$ we obtain the model

Maximise

$$z \tag{3.116}$$

subject to

$$z \leq -3u_1 + 2u_2 \tag{3.117}$$
$$z \leqslant 4v_1 - 2v_2 + 2v_3 \tag{3.118}$$
$$z \leqslant w_2 + 3w_3 \tag{3.119}$$
$$2u_1 - u_2 + u_3 \leqslant 1 \tag{3.120}$$
$$-u_1 + u_2 \leqslant 3 \tag{3.121}$$
$$4v_1 - 2v_2 + v_3 \leqslant 1 \tag{3.122}$$
$$-2v_1 + 2v_2 + v_3 \leqslant 3 \tag{3.123}$$
$$6w_1 - 3w_2 + 2w_3 \leqslant 1 \tag{3.124}$$
$$-3w_1 + 3w_2 + 3w_3 \leqslant 3 \tag{3.125}$$
$$u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3 \geqslant 0 \tag{3.126}$$

$$u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3 \in \mathbb{R}$$

Using the notation introduced in Sect. 3.3 we can write the above model as follows:

Maximise

$$(-3u_1 + 2u_2) \oplus (4v_1 - 2v_2 + 2v_3) \oplus (w_2 + 3w_3) \tag{3.127}$$

subject to

$$(2u_1 - u_2 + u_3) \oplus' (4v_1 - 2v_2 + v_3) \oplus' (6w_1 - 3w_2 + 2w_3) \leqslant 1 \tag{3.128}$$

$$(-u_1 + u_2) \oplus' (-2v_1 + 2v_2 + v_3) \oplus' (-3w_1 + 3w_2 + 3w_3) \leqslant 3 \tag{3.129}$$

$$u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3 \geqslant 0 \tag{3.130}$$

$$u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3 \in \mathbb{R}$$

This is obviously the dual of the disjunctive programme stated in Example 3.5. It therefore has the same optimal objective value (so long as both models are feasible). Hence the LP dual of the model above, namely the LP relaxation of the MIP model created in Example 3.10, is an LP representation of the disjunctive programme in Example 3.5, i.e. our 'disjunctive' formulation (by splitting variables) is sharp. This demonstration is entirely general and applies to any disjunctive programme. We have represented the convex hull of the disjunctive formulation (in a higher dimensional space). It is possible (but often of exponential complexity) to project this formulation back into the space of the original variables (e.g. using Fourier–Motzkin elimination, as described in Chapter 2).

We should also point out that the possibility of this type of formulation does not affect the issue of MIP representability, discussed in Section 3.2. This type of formulation is possible if and only if the problem is MIP representable.

3.5 Modelling Languages Based On Logic

3.5.1 Algebraic Languages

The use of modelling languages to aid the building of LP and IP models has now become widespread. These languages typically use *index sets* to refer to variables and constraints in the model, so allowing classes of variables and constraints to be referred to once. They also allow the *structure* of the model to be separated from the *data.* It then becomes possible to generate variables, constraints and objectives conditional on the data, e.g. if particular numerical conditions hold or entities are members of particular sets. In order to do this logic, in the form of the propositional calculus, is usually incorporated into the language.

Full details of any one language are specific to the software system used and described in the appropriate manual. However, we illustrate typical logical facilities

by means of the commonly used language AMPL (**A** **M**athematical **P**rogramming **L**anguage).

This language allows data to be defined in terms of numerical scalars, vectors and arrows as well as index sets. It is then possible to make statements about the data which may be $true$ or $false$. Compound propositions can then be constructed from these statements. In AMPL the possible 'connectives' which may be used are

$$or, not, and, if \ldots then \ldots else, exits, forall \tag{3.131}$$

obviously representing '$\vee$','$-$','$\cdot$','$\longrightarrow$','$\exists$','$\forall$', respectively.

A typical statement in AMPL might be

$$\begin{aligned} & & subject\ to & \\ Capacity\{i\ in\ MACHINES\} & : & sum(j\ in\ PRODUCTS) & \\ & & Time[i,j] * x[i,j] & \\ & < = & (exists\ (Time[i,j] > 0) & \\ & & and\ not\{i\ in\ PRECLUDE\}) & \\ & & Cap[i] & \end{aligned} \tag{3.132}$$

This statement generates a number of constraints indexed by the set $MACHINES$. The number of each product (indexed by $PRODUCTS$) produced is represented by an LP (or IP) variable $x[i, j]$. Each variable has a coefficient $Time[i, j]$ taken from a data array. The capacity (in hours) of each machine is given in the data vector $Cap[i]$.

However, the constraint will only be generated if the following statement is true:

$$exists\ (Time[i,j] > 0)\ and\ not\{i\ in\ PRECLUDE\} \tag{3.133}$$

i.e. **both** at least one of the products takes a positive time on the relevant machine **and** the machine is **not** one of a precluded set for which no capacity constraint is needed.

(Note that the names of the sets and data arrays have been chosen to make the statement meaningful. They have no significance in the language.)

This language is typical of the 'state of the art' in such algebraic languages. It is also possible, in some languages, to generate constraints using the rich language of *constraint logic programming*, as described in Chapter 4.

We now describe a language and method of generating IP models which goes beyond what can be achieved by purely algebraic modelling languages.

3.5.2 The 'Greater Than or Equal' Predicate

We define the predicate $ge(r, P_1, P_2, \ldots, P_n)$ as meaning greater than or equal to r of the statements $P_1, P_2, \ldots, P_n$ are true. (This predicate could equally well have been called 'at-least'.) This predicate is itself a statement which will be true or false. (Obviously if $r > n$ it must be false.)

If $r = 1$ it represents the disjunction

$$P_1 \vee P_2 \vee \cdots \vee P_n \tag{3.134}$$

and if $r = n$ it represents the conjunction

$$P_1 \cdot P_2 \cdot \cdots \cdot P_n \tag{3.135}$$

By taking values of r between 1 and n we can represent compound statements between (multivalued) '$\vee$' and '$\cdot$'. Such statements could, of course, be written in terms of the conventional connectives but such representations would be more cumbersome.

What's more the statements using 'ge' can be *nested*, i.e. the component statements P_i can themselves be ge predicates.

We illustrate this by an example

Example 3.12 Represent the following statement in DNF using conventional connectives:

$$ge(2, ge(1, P_1, P_2), P_3, ge(2, P_4, P_5)) \tag{3.136}$$

we can progressively reexpress (3.136) (from 'the bottom up') as follows:

$$ge(2, P_1 \vee P_2, P_3, P_4 \cdot P_5) \tag{3.137}$$

$$(P_1 \vee P_2) \cdot P_3 \vee (P_1 \vee P_2) \cdot P_4 \cdot P_5) \vee P_3 \cdot P_4 \cdot P_5 \tag{3.138}$$

$$P_1 \cdot P_3 \vee P_2 \cdot P_3 \vee P_1 \cdot P_4 \cdot P_5 \vee P_2 \cdot P_4 \cdot P_5 \vee P_3 \cdot P_4 \cdot P_5 \tag{3.139}$$

At the bottom level the component statements will be LP or IP constraints or 0–1 variables which can individually be true or false. The representation entirely in terms of nested ge predicates gives a 'normal form' (although it is not unique) into which all models can be put. The translation into an IP model can then be automated. We describe how this may be done below. Before doing this, however, we point out that nested ge predicates are not a natural way of modelling a problem. Instead we use a richer set of predicates, together with the conventional connectives of the propositional calculus. Then we translate into the ge form before translating into an IP model.

Useful predicates which we use (in addition to the conventional connectives and ge itself) are as follows:

$le(r, P_1, P_2, \ldots, P_n)$ meaning less than or equal to r of $P_1, P_2, \ldots, P_n$ are true.
$eq(r, P_1, P_2, \ldots, P_n)$ meaning exactly r of $P_1, P_2, \ldots, P_n$ are true.
$gef(r, P_1, P_2, \ldots, P_n)$ meaning greater than or equal to r of $P_1, P_2, \ldots, P_n$ are false.
$lef(r, P_1, P_2, \ldots, P_n)$ meaning less than or equal to r of $P_1, P_2, \ldots, P_n$ are false.
$eqf(r, P_1, P_2, \ldots, P_n)$ meaning exactly r of $P_1, P_2, \ldots, P_n$ are false.
$le(r, P_1, P_2, \ldots, P_n)$ meaning less than or equal to r of $P_1, P_2, \ldots, P_n$ are true.
$at\_least(r, P_1, P_2, \ldots, P_n)$ meaning at least r of $P_1, P_2, \ldots, P_n$ are true, i.e. $ge(r, P_1, P_2, \ldots, P_n)$.
$at\_most(r, P_1, P_2, \ldots, P_n)$ meaning at most r of $P_1, P_2, \ldots, P_n$ are true, i.e. $le(r, P_1, P_2, \ldots, P_n)$.
$all(P_1, P_2, \ldots, P_n)$ meaning all of $P_1, P_2, \ldots, P_n$ are true, i.e. $ge(n, P_1, P_2, \ldots, P_n)$.
$none(P_1, P_2, \ldots, P_n)$ meaning none of $P_1, P_2, \ldots, P_n$ are true, i.e. $ge(0, P_1, P_2, \ldots, P_n)$.

These predicates are for modelling purposes. Ultimately they will be converted (automatically) into the ge form, within the system, by rules given below. In addition we may wish to use the rich variety of predicates available from *constraint logic programming*, as described in Chapter 4.

The conversion of the above predicates to ge form is straightforward and left as Exercise 3.7.7. For illustration, however, we consider the le predicate $le(r, P_1, P_2, \ldots, P_n)$ converts to $ge(r + 1, \overline{P}_1, \overline{P}_2, \ldots, \overline{P}_n)$.

Negations of a ge can always be moved inwards (a generalisation of De Morgan's laws), i.e. $\tilde{}ge(r, P_1, P_2, \ldots, P_n)$ converts to $ge(n - r + 1, \overline{P}_1, \overline{P}_2, \ldots, \overline{P}_n)$.

Models resulting from this approach to IP formulation may, sometimes, be simplified. We point out two obvious simplifications here which we refer to as 'flattening'

$$ge(1, ge(1, P_1, P_2, \ldots, P_m), P_{m+1}, P_{m+2}, \ldots, P_n) \tag{3.140}$$

can be rewritten as

$$ge(1, P_1, P_2, \ldots, P_n) \tag{3.141}$$

and

$$ge(n + 1, ge(m, P_1, P_2, \ldots, P_m), P_{m+1}, P_{m+2}, \ldots, P_n) \tag{3.142}$$

can be rewritten as

$$ge(n, P_1, P_2, \ldots, P_n) \tag{3.143}$$

so removing one level of nesting in each case.

These two simplifications are the result of the associativities of '$\vee$' and '$\cdot$', respectively.

We demonstrate how to use the 'ge' predicate to create an IP model by the following example.

Example 3.13 Model the following condition among constraints by a MIP model:

$$\sum_j a_{1j}x_j \leqslant b_1 \vee \sum_j a_{2j}x_j \geqslant b_2 \leftrightarrow \sum_j a_{3j}x_j \geqslant b_3 \vee \sum_j a_{4j}x_j \leqslant b_4 \quad (3.144)$$

It is convenient to represent the four inequalities by P_1, P_2, P_3, P_4. At the 'top level' we model the '$\leftrightarrow$' by

$$ge(2, (P_1 \vee P_2) \rightarrow (P_3 \vee P_4), (P_3 \vee P_4) \rightarrow (P_1 \vee P_2)) \quad (3.145)$$

Proceeding down the component statements give

$$\begin{aligned} ge(2,\, & ge(1, \tilde{}(P_1 \vee P_2), (P_3 \vee P_4)), \\ & ge(1, \tilde{}(P_3 \vee P_4), (P_1 \vee P_2))) \end{aligned} \quad (3.146)$$

$$\begin{aligned} ge(2,\, & ge(1,\, ge(2,(\overline{P}_1,\, \overline{P}_2)), \\ & \quad ge(1,\, P_3,\, P_4)), \\ & ge(1,\, ge(2,\, (\overline{P}_3,\, \overline{P}_4), \\ & \quad ge(1,\, P_1,\, P_2)))) \end{aligned} \quad (3.147)$$

By flattening we can simplify (3.147) to

$$\begin{aligned} ge(2,\, & ge(1,\, ge(2,\, (\overline{P}_1, \overline{P}_2)), \\ & \quad P_3,\, P_4), \\ & ge(1,\, ge(2,\, (\overline{P}_3, \overline{P}_4)), \\ & \quad P_1,\, P_2)) \end{aligned} \quad (3.148)$$

At the bottom level we introduce variables δ_1, δ_2, δ_3, δ_4 and model the conditions

$$\delta_i = 1 \leftrightarrow P_i \quad (3.149)$$

Assuming suitable bounds are known for the expressions in the inequalities then we can model these conditions by the methods given in Sect. 3.1. Otherwise we can use the method for open polyhedra described in Sect. 3.2 (so long as the problem is MIP representable). Assuming the bounds are known this gives

$$\sum_j a_{1j}x_j + (M_1 - b_1)\delta_1 \leqslant M_1 \tag{3.150}$$

$$\sum_j a_{2j}x_j + (m_2 - b_2)\delta_2 \geqslant m_2 \tag{3.151}$$

$$\sum_j a_{3j}x_j + (m_3 - b_3)\delta_3 \geqslant m_3 \tag{3.152}$$

$$\sum_j a_{4j}x_j + (M_4 - b_4)\delta_4 \leqslant M_4 \tag{3.153}$$

Since the predicates $ge(2, (\overline{P}_1, \overline{P}_2))$ and $ge(2, (\overline{P}_3, \overline{P}_4))$ are of the form (3.135) (conjunctions) each can be represented by single 0–1 variables using the conditions

$$\delta_5 = 0 \leftrightarrow \overline{P}_1 \tag{3.154}$$

$$\delta_5 = 0 \leftrightarrow \overline{P}_2$$

$$\delta_6 = 0 \leftrightarrow \overline{P}_3 \tag{3.155}$$

$$\delta_6 = 0 \leftrightarrow \overline{P}_4$$

giving

$$\sum_j a_{1j}x_j - (m_1 - b_1 + \varepsilon)\delta_5 \geqslant b_1 + \varepsilon \tag{3.156}$$

$$\sum_j a_{2j}x_j - (M_2 - b_2 + \varepsilon)\delta_5 \leqslant b_2 - \varepsilon \tag{3.157}$$

$$\sum_j a_{3j}x_j - (M_3 - b_3 + \varepsilon)\delta_6 \leqslant b_3 - \varepsilon \tag{3.158}$$

$$\sum_j a_{4j}x_j - (m_4 - b_4 + \varepsilon)\delta_6 \geqslant b_4 + \varepsilon \tag{3.159}$$

Moving up one level we model

$$ge(1, ge(2, (\overline{P}_1, \overline{P}_2)),\ P_3, P_4) \tag{3.160}$$

by

$$\delta_3 + \delta_4 - \delta_5 \geqslant 0 \tag{3.161}$$

and

$$ge(1, ge(2, (\overline{P}_3, \overline{P}_4)), P_1, P_2) \tag{3.162}$$

by

$$\delta_1 + \delta_2 - \delta_6 \geqslant 0 \tag{3.163}$$

No 0–1 variable is necessary at the top level since we have a predicate of the form (3.135). Therefore the full condition (3.144) is modelled by (3.150) to (3.153) together with (3.156) to (3.159) and (3.161) and (3.163).

3.6 References and Further Work

Williams [117] presents the relationship between connectives in the propositional calculus and IP constraints. This is also covered in Williams [112, 115, 119]. Other authors have also written on the subject such as Barth [12], Hürlimann [58], Dietricht et al. [33], Chandru and Hooker [22], Hooker [54, 56] and Wilson [121, 122] .

Balas [5, 7, 9] was the first to develop the theory of disjunctive programming and the resultant cutting planes in [6]. Sherali and Shetty [100] also cover the subject.

The theory of MIP representability was developed by Meyer [82] and Jeroslow [62, 63]. Jeroslow also (stemming from the work of Balas) produced the idea of the splitting of variables to lead to convex formulations (under the slightly misleading title 'convex formulations'). Jeroslow and Lowe [64] describe modelling techniques based on these methods. Martin[78] obtains a number of IP reformulations by means of new variables.

Improved (tighter) IP formulations, by means of splitting variables, exist for a number of applications. Sometimes it is possible to (fictitiously) introduce 'multiple commodities' into a problem where there is only one commodity to increase tightness. Wolsey [126] does this for the lot-sizing problem and Wong [127] and Claus [25] for the network flow formulation of the travelling salesman problem.

Williams [116] gives the duality-based proof to show that the convex formulation is always sharp. Minimax algebra (using the operators '$\oplus$' and '$\oplus'$') was developed by Carré [21] and Cunningham Green [28].

Modelling languages have been developed by a number of authors such as Fourer et al. [37] for the AMPL system and Day and Williams [32]. McKinnon and Williams [80] developed the modelling system based on the recursive use of the 'greater-than-or-equal-to' predicate. They also show how to avoid introducing a component for every variable in every disjunction. Hadjiconstantinou et al. [50] present another way of systematically converting a logical expression into an IP model. Greenberg and Murphy [48] compare different modelling systems.

3.7 Exercises

3.7.1 Model the disjunction of r constraints using $\lceil \log_2 r \rceil$ 0–1 variables as described in Sect. 3.1. Is the LP relaxation weaker than that for the formulation using r constraints?

3.7.2 Show that if the open facets of a disjunction of polyhedra contain at least as many extreme rays as the number of variables then they represent parallel facets.

3.7.3 Give an example of a set of polyhedra with the same recession directions whose open facets are not parallel. Give a MIP representation of the disjunction of these polyhedra without splitting the variables.

3.7.4 Show that statement (3.54) is equivalent to (3.55).

3.7.5 Reformulate Example 3.5 in CNF and solve the resultant IP.

3.7.6 Formulate the dual of Example 3.5.

3.7.7 Show that the dual of the dual of a disjunctive programme is the primal.

3.7.8 Convert the predicates in Sect. 3.5 into 'ge' form.

3.7.9 Solve Example 3.2 with objective (3.20) using the method described in Example 2.3.

3.7.10 Solve Example 3.5 using the method described in Example 2.3.

Chapter 4
The Satisfiability Problem and Its Extensions

Given a statement in logic can it ever be true? This is the *satisfiability* problem. We will confine our attention to the propositional calculus. However, the problem also arises in the predicate calculus where it is necessary to consider if there are instantiations of the variables which make a statement true. The problem is equivalent to the inference and consistency problems mentioned in Chapter 1.

Suppose we want to decide if

$$P \Longrightarrow Q \tag{4.1}$$

i.e. can we *infer* Q from P, where P and Q are logical statements or sets of logical statements. Equation (4.1) is equivalent to the logical statement

$$P \longrightarrow Q \tag{4.2}$$

(which could also be written as $\overline{P} \vee Q$).

We simply need to test if (4.2) is always true, i.e. if its negation can never be true. The negation can be represented by

$$P \cdot \overline{Q} \tag{4.3}$$

using De Morgan's laws.

Hence is (4.3) **ever** true?, i.e. is it satisfiable? If it is then the inference (4.1) does not hold, otherwise it does hold.

Conversely if we have a satisfiability problem of testing if a statement, or set of statements, P is satisfiable we can (trivially) convert it to the inference problem

$$P \Longrightarrow \mathbf{F} \tag{4.4}$$

If this inference is **not** true then P is sometimes true, i.e. it is satisfiable.

If we wish to check if a set of statements

$$P_1, P_2, P_3, \ldots, P_n \tag{4.5}$$

H.P. Williams, *Logic and Integer Programming,* International Series in Operations Research & Management Science 130, DOI 10.1007/978-0-387-92280-5_4,
© Springer Science+Business Media, LLC 2009

is consistent then we simply need to test if the statement

$$P_1 \cdot P_2 \cdot P_3 \cdots P_n \tag{4.6}$$

is satisfiable.

Conversely saying P is satisfiable is the same as saying it is consistent.

Satisfiability problems with a small number of literals could be solved by the use of **truth tables** as follows.

Example 4.1 Is the following statement satisfiable?

$$(X_1 \longrightarrow (X_2 \vee X_3)) \rightarrow (\overline{X_1} \cdot \overline{X_3}) \tag{4.7}$$

We can successively fill in the columns of the following truth table (Table 4.1)

Table 4.1 A Truth Table for Satisfiability

X_1	X_2	X_3	$X_2 \vee X_3$	$X_1 \longrightarrow X_2 \vee X_3$	$\overline{X_1} \cdot \overline{X_3}$	$(X_1 \longrightarrow X_2 \vee X_3) \longrightarrow (\overline{X_1} \cdot \overline{X_3})$
T	T	T	T	T	F	F
T	T	F	T	T	F	F
T	F	T	T	T	F	F
T	F	F	F	F	F	T
F	T	T	T	T	F	F
F	T	F	T	T	T	T
F	F	T	T	T	F	F
F	F	F	F	T	T	T

This demonstrates that the statement is satisfiable. It is satisfiable for the three truth table settings

$$X_1 = \mathbf{T}, \quad X_2 = \ X_3 = \mathbf{F}$$
$$X_1 = \mathbf{F}, \quad X_2 = \mathbf{T}, \quad X_3 = \mathbf{F}$$
$$X_1 = X_2 = X_3 = \mathbf{F}$$

It can be seen that the number of rows in such a truth table will be 2^n, where n is the number of literals. 2^n grows exponentially rendering such an approach impractical for other than small problems. We present practical alternatives (although in the 'worst-case' satisfiability problems are of exponential complexity, as discussed in Section 2.4).

4.1 Resolution and Absorption

In order to carry out a more efficient procedure we write a logical statement in conjunctive normal form (CNF). Therefore we consider statements written as a conjunction of disjunctive clauses. We will use the following, larger, example.

Example 4.2 Is the following statement satisfiable?

$$\overline{X_1} \vee X_3 \vee X_4 \tag{4.8}$$
$$\cdot\overline{X_2} \vee X_3 \vee X_4 \tag{4.9}$$
$$\cdot X_1 \vee \overline{X}_3 \tag{4.10}$$
$$\cdot\overline{X_3} \vee X_4 \tag{4.11}$$
$$\cdot\overline{X_1} \vee X_3 \vee \overline{X_4} \tag{4.12}$$
$$\cdot\overline{X}_1 \vee \overline{X_2} \tag{4.13}$$
$$\cdot\overline{X_2} \vee X_3 \tag{4.14}$$
$$\cdot X_2 \vee X_4 \tag{4.15}$$
$$\cdot X_3 \vee X_4 \tag{4.16}$$

i.e. we have a conjunction of the (disjunctive) clauses which are written on each line.

If it turns out that a conjunction of clauses, such as that above, is not satisfiable then there is also interest in finding the maximum number of clauses which are, together, satisfiable. This is known as the *maximum satisfiability* problem. We will show how to solve it as an IP model in Sect. 4.5.

Before solving Example 4.2, in Sect. 4.2, we define two procedures needed.

Resolution consists of choosing two clauses which have *exactly one* literal which is negated in one clause and unnegated in the other (they may or may not have other literals in common, but if so they must have the same sign).

An example is provided by clauses (4.9) and (4.12).

$$\overline{X_2} \vee X_3 \vee X_4 \quad \text{and} \quad \overline{X_1} \vee X_3 \vee \overline{X_4} \tag{4.17}$$

Two such clauses can be combined by forming their disjunction and deleting the common literal (in this case X_4) which is negated in one and unnegated in the other. The resultant clause is referred to as their *resolvent.* In the case of (4.17) we obtain the resolvent

$$\overline{X_1} \vee \overline{X_2} \vee X_3 \tag{4.18}$$

Clearly the resolvent is implied by the original two clauses since the eliminated common literal (X_4) is either true or false. If it is true the remaining part of one of the clauses is true and if it is false the remaining part of the other clause is true. Hence either way the *disjunction* of them is true giving the resolvent, i.e. the resolvent is an *implication* of the original statement.

Absorption involves looking for clauses that contain all the literals of another clause and the common literals have the same sign. The smaller clause is said to *absorb* the larger. For example (4.16) absorbs (4.8). Clearly the smaller clause implies the larger which may therefore be removed (obviously if there are duplicate clauses either one can be removed).

To simplify a logical statement in CNF we repeatedly apply absorption and resolution, appending the resolvent to the set of clauses and removing absorbed clauses, until no further simplification is possible. The result is a '*simpler* ' but equivalent statement.

Applying these procedures to the example we finally obtain the conjunction

$$\overline{X_2} \cdot X_4 \cdot (\overline{X_1} \vee X_3) \cdot (X_1 \vee \overline{X_3}) \tag{4.19}$$

The individual clauses obtained (in this example $\overline{X_2}$, X_4, $\overline{X_1} \vee X_3$, $X_1 \vee \overline{X}_3$) are known as *prime implications*. They are the smallest (i.e. least number of literals) disjunctive clauses that are implied by the original statement.

This procedure delivers us the *complete sum of prime implications*, i.e. every possible minimal clausal prime implication. All other clausal implications would be absorbed by a prime implication (and therefore not be prime).

In order to show this suppose we were to have a disjunctive clause C which is implied by the original statement and is the longest clause not absorbed by any of the prime implications. We can, without loss of generality, assume that all the literals in C are unnegated (by, if necessary, replacing the negated literals throughout by unnegated ones). Therefore let

$$C = X_1 \vee X_2 \vee \cdots \vee X_r \tag{4.20}$$

In order for C to be false $X_1 = X_2 = \cdots = X_r = \mathbf{F}$. Therefore C cannot contain all the literals in the original statement as the only way for the original statement to be false would be for all the literals in the prime implications to be unnegated. This would cause them each to absorb C. Therefore C must not contain some variable, say X_i. In this case, since C is the longest clause not absorbed $X_i \vee C$ and $\overline{X_i} \vee C$ which are both implied must be absorbed by prime implications. The prime implication which absorbs $X_i \vee C$ must contain X_i otherwise it would absorb C. The prime implication which absorbs $\overline{X_i} \vee C$ must contain $\overline{X_i}$ for a similar reason. The resolvent of these two implications would be a prime implication which absorbs C contradicting the definition of C.

The complete sum of prime implications is not, however, necessarily the *simplest* equivalent statement. There might be a proper subset of the prime implications, the conjunction of which is still an equivalent statement. The finding of the minimal equivalent set of prime implications is, however, another (difficult) problem. We address this problem in Sect. 4.6. For the small example above, the four derived prime implications also form the minimal set. However, we will now use resolution and absorption, together with a branching procedure, for the more modest aim of determining if a statement in CNF is satisfiable.

4.2 The Davis–Putnam–Loveland (DPL) Procedure

If a statement is not satisfiable then the empty clause is a prime implication (which absorbs all other implications rendering it the only one) and this will be obtained by successively applying resolution and absorption. Otherwise, if the empty clause is not obtained, then the statement is satisfiable. A satisfiable set of truth values can be obtained as follows:

1. (a) If a clause consists of a single literal X then set $X = \mathbf{T}$. Similarly if a clause consists of a single literal $\overline{X}$ set $X = \mathbf{F}$. Note: There will be no other clauses containing X or $\overline{X}$ by the application of absorption.
   (b) If a literal is unnegated in all clauses within which it occurs set $X = \mathbf{T}$ and delete the clauses within which it occurs. (There may be alternative satisfying settings but this is sufficient if we are content only to find a satisfiable setting.) Similarly if a literal is negated in all clauses within which it occurs set $X = \mathbf{F}$ and delete all clauses within which it occurs. (Again there may be alternatives.)
   (c) If a literal X is unnegated in some clauses and negated in others partition the clauses into two sets (i) and (ii).

   (i) Set $X = \mathbf{T}$ and delete the clauses in which it is unnegated.
   (ii) Set $X = \mathbf{F}$ and delete the clauses in which it is negated.

   Each of the sets (i) and (ii) after this 'branching' is then treated as before by procedures (a), (b) and (c). It can be shown (Exercise 4.12.3) that at least one of the branches will result in a satisfiable set of truth values.

For, comparatively simple, Example 4.2 resolution and absorption resulted in (4.19): (a) sets $X_2 = \mathbf{F}$, $X_4 = \mathbf{T}$; (b) does not apply and (c) creates the settings (branching on, say, X_1): (i) $X_1 = \mathbf{T}$, $X_3 = \mathbf{T}$ and (ii) $X_1 = \mathbf{F}$, $X_3 = \mathbf{F}$.

After applying (b) we obtain the alternative sets of satisfying truth values

$$X_1 = \mathbf{T}, \quad X_2 = \mathbf{F}, \quad X_3 = \mathbf{T}, \quad X_4 = \mathbf{T} \tag{4.21}$$

$$X_1 = \mathbf{F}, \quad X_2 = \mathbf{F}, \quad X_3 = \mathbf{F}, \quad X_4 = \mathbf{T} \tag{4.22}$$

4.3 Representation as an Integer Programme

The satisfiability problem can be represented as the problem of deciding if an integer programme (IP) is feasible using the modelling methods described in Chapter 3. However, the satisfiability problem has a special structure as an IP which should be explained.

If the satisfiability problem is expressed in CNF then each (disjunctive) clause gives rise to a constraint. As in Chapter 2 we represent the truth or falsity of a literal

X by a 0–1 variable x taking the value 1 or 0, respectively. If X occurs unnegated in a clause then x appears in the constraint. If it occurs negated then $(1 - x)$ appears. The sum of these terms in the constraint must be $\geqslant 1$.

For example the clause

$$\overline{X_1} \vee X_3 \vee X_4 \tag{4.23}$$

gives rise to the constraint

$$(1 - x_1) + x_3 + x_4 \geqslant 1 \tag{4.24}$$

In more usual form this is written as

$$-x_1 + x_3 + x_4 \geqslant 0 \tag{4.25}$$

For Example 4.2 the full set of constraints is therefore

$$-x_1 + x_3 + x_4 \geqslant 0 \tag{4.26}$$
$$-x_2 + x_3 + x_4 \geqslant 0 \tag{4.27}$$
$$x_1 - x_3 \geqslant 0 \tag{4.28}$$
$$-x_3 + x_4 \geqslant 0 \tag{4.29}$$
$$-x_1 + x_3 - x_4 \geqslant -1 \tag{4.30}$$
$$-x_1 - x_2 \geqslant -1 \tag{4.31}$$
$$-x_2 + x_3 \geqslant 0 \tag{4.32}$$
$$x_2 + x_4 \geqslant 1 \tag{4.33}$$
$$x_3 + x_4 \geqslant 1 \tag{4.34}$$

The general form of such constraints is

$$-\sum_{i=1}^{r} x_i + \sum_{i=r+1}^{r+s} x_i \geqslant (1 - r) \tag{4.35}$$

Models with such constraints are a special case of generalised set-covering problems (GSCPs). (GSCPs, in contrast to the set-covering problem mentioned in Chapter 2, have all coefficients 0, ± 1 and general integer right-hand sides, with all constraints '$\geqslant$'.) This special sort of GSCP can easily be converted to a set-covering problem (SCP) where all the coefficients and right-hand sides are 0–1 by substituting $(1 - y_i)$ for x_i in the first summation in (4.35). This then gives the constraints

$$\sum_{i=1}^{r} y_i + \sum_{i=r+1}^{r+s} x_i \geqslant 1 \tag{4.36}$$

Together with the extra constraints

$$x_i + y_i = 1, \quad i = 1, \ldots, r \tag{4.37}$$

If (4.37) were converted to the '$\geqslant$' form then solutions satisfying the '=' form could be sought by the choice of suitable objective function such as

$$\text{Minimise} \quad \sum_{i=1}^{r} (x_i + y_i) \tag{4.38}$$

If this objective can be minimised to r then (4.37) must be satisfied and the original problem is satisfiable.

However, for our purposes it suffices to consider satisfiability problems in the form of GSCPs.

4.4 The Relationship Between Resolution and Cutting Planes

We could solve the problems described above as IPs using the branch-and-bound method described in Chapter 2. However, it is worth pointing out that we can often, with benefit, **reduce** the size of such models first. The methods of reduction we use mirror *resolution* and *absorption* as described in Sect. 4.1. We illustrate this by reference to constraints (4.8) and (4.16).

If one constraint is implied by another it can be removed. This happens when one contains a subset of the variables of another with the same signs on the variables and a relaxation of the right-hand side. For example

$$x_3 + x_4 \geqslant 1 \quad \text{clearly implies} \quad -x_1 + x_3 + x_4 \geqslant 0 \tag{4.39}$$

i.e. (4.16) implies (4.8). This mirrors the absorption of $\overline{X_1} \vee X_3 \vee X_4$ by $X_3 \vee X_4$.

Corresponding to the *resolution* operation we can *add* together two constraints which contain a variable which is negated in one constraint and not in the other. If all the other common variables have the same sign the resultant constraint is of significance. Consider the two constraints (4.27) and (4.30). Adding them together produces

$$-x_1 - x_2 + 2x_3 \geqslant -1 \tag{4.40}$$

There is also no loss of generality in adding in the conditions (in negated form) $-x_1 \geqslant -1$ and $-x_2 \geqslant -1$ in order to make all the coefficients ± 2 giving

$$-2x_1 - 2x_2 + 2x_3 \geqslant -3 \tag{4.41}$$

Since the quantity on the left-hand side is a multiple of 2 we can divide through by 2 (this is one of the operations for creating the Chvátal 'dual' as described in Chapter 2) to produce

$$-x_1 - x_2 + x_3 \geqslant -\frac{3}{2} \tag{4.42}$$

and round up the right-hand side to give

$$-x_1 - x_2 + x_3 \geqslant -1 \tag{4.43}$$

This procedure mirrors resolution applied to $\overline{X_2} \vee X_3 \vee X_4$ and $\overline{X_1} \vee X_3 \vee \overline{X_4}$ to produce $\overline{X_1} \vee \overline{X_2} \vee X_3$ which is represented by the constraint (4.43).

The above procedure, applied to constraints, is very significant. Not only we are adding constraints to produce a new constraint but we are also applying a **rounding** operation. The result of this procedure is a **Rank 1 Cutting Plane** as explained in Chapter 2. It therefore cuts off solutions to the linear programming (LP) relaxation but does not cut off any integer solutions. Hence the *reduced* model may well have a tighter LP relaxation than that of the original model. This, almost certainly, makes the subsequent solution by branch-and-bound shorter.

It might be thought that there is an exact relationship between prime implications (the smallest implied clauses) and IP facets (the tightest constraints) as discussed in Chapter 2. This is not, however, the case (see Exercise 4.12.9).

Exercise 4.12.7 involves showing that, in general, the procedure above exactly mirrors resolution. Exercise 4.12.8 involves showing that if one adds constraints with more than variable differing in sign no rounding (and hence strengthening) applies (although the resultant constraint is still valid).

We now apply branch-and-bound to the set of constraints which results from reducing (4.26) to (4.34) by using the constraint forms of resolution and absorption. This corresponds to the logical statement (3.19) which was obtained after logical resolution and absorption applied to the original, logical, statements (3.8)–(3.16). The corresponding set of reduced constraints is

$$-x_2 \geqslant 0, \quad x_4 \geqslant 1, \quad -x_1 + x_3 \geqslant 0, \quad x_1 - x_3 \geqslant 0 \tag{4.44}$$

Together with the restriction that these variables be 0–1 and the (arbitrary) objective

$$\text{Minimise} \quad x_1 + x_2 + x_3 + x_4 \tag{4.45}$$

we obtain the LP relaxation solution

$$x_1 = 0, \quad x_2 = 0, \quad x_3 = 0, \quad x_4 = 1 \tag{4.46}$$

This is clearly integral in this, very simple, case demonstrating that the original (logical) statement was satisfiable. In general we may obtain a fractional solution and have to resort to branch-and-bound.

Notice that by choice of objective (4.45) we have produced only one of the two solutions corresponding to (4.22). The choice of objective can be arbitrary if we are only seeking to prove *satisfiability.* If we require *all* satisfying solutions then we have the much more difficult task of finding *all* feasible solutions to the IP. We do not address this task here.

4.5 The Maximum Satisfiability Problem

In order to illustrate further the use of IP we consider an important extension of the satisfiability problem and illustrate it by an example.

Example 4.3 Is the following statement satisfiable and if not what is the maximum number of clauses which make it satisfiable?

$$X_{11} \vee X_{12} \tag{4.47}$$
$$\cdot\ X_{21} \vee X_{22} \tag{4.48}$$
$$\cdot\ X_{31} \vee X_{32} \tag{4.49}$$
$$\cdot\ \overline{X_{11}} \vee \overline{X_{21}} \tag{4.50}$$
$$\cdot\ \overline{X_{11}} \vee \overline{X_{31}} \tag{4.51}$$
$$\cdot\ \overline{X_{21}} \vee \overline{X_{31}} \tag{4.52}$$
$$\cdot\ \overline{X_{12}} \vee \overline{X_{22}} \tag{4.53}$$
$$\cdot\ \overline{X_{12}} \vee \overline{X_{32}} \tag{4.54}$$
$$\cdot\ \overline{X_{22}} \vee \overline{X_{32}} \tag{4.55}$$

This is a tiny example of the so-called 'pigeon hole' problem. This is the problem of seeing if it is possible to fit $n+1$ objects into n boxes with no more than one object in each box. Clearly this is not possible when looked at *semantically*. However, to prove that it is not possible *syntactically* (if we did not know the interpretation) is very difficult. It can be shown (Exercise 4.12.10) that, for the general problem, the number of steps needed to do this by resolution is an exponential function of n.

For the above example we interpret X_{ij} as meaning 'object i is put in box j'. Clauses (4.47)–(4.49) impose the condition that each i must be put somewhere. Clauses (4.50)–(4.55) impose the condition that no more than one object can be put in each box j. Here we have three objects and two boxes.

It is convenient to illustrate our application of resolution by means of the tree in Fig. 4.1.

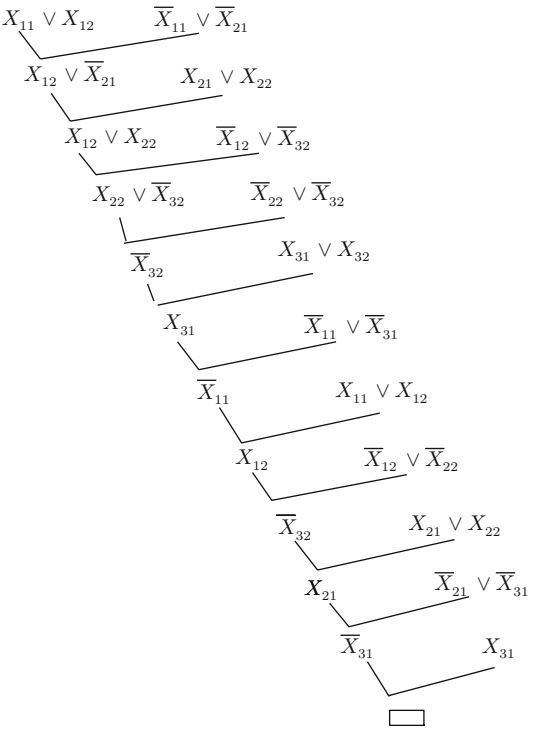


Fig. 4.1 A resolution tree

At each level we apply *resolution*. For example at level 1 we resolve clauses (4.47) and (4.50) to obtain $X_{12} \vee \overline{X_{21}}$ which is then resolved with (4.48). Proceeding in this way we ultimately resolve $\overline{X_{31}}$ with X_{31} to produce the empty clause represented by '$\square$' showing that the original statement is not satisfiable.

Note that at the end both $\overline{X_{31}}$ and X_{31} are 'non-input' clauses, having been derived in the course of the calculation. This is of importance in distinguishing between 'easy' and 'difficult' problems and is addressed in Sect. 4.6 and in Exercise 4.12.11.

Given that the statement in Example 4.3 is not satisfiable, what is the maximum number of clauses which can simultaneously be made satisfiable? This is an example of the *maximum satisfiability problem.*

If we represent a general (disjunctive) clause as a constraint among 0–1 variables in the form of (4.35) (indexed as constraint j) we can force its satisfaction by a 0–1 variable y_j taking the value 1. In order to do this we write the constraint as (4.56) using the modelling methods discussed in Chapter 3.

$$-\sum_{i=1}^{r} x_i + \sum_{i=r+1}^{r+s} x_i - y_j \geqslant -r \tag{4.56}$$

By employing the objective

$$\text{Maximise} \quad \sum_j y_j \tag{4.57}$$

we maximise the number of clauses which are satisfied.

We apply this model to Example 4.3 to give Maximise

$$y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_9 + y_9 \tag{4.58}$$

subject to

$$x_{11} + x_{12} - y_1 \geqslant 0 \tag{4.59}$$
$$x_{21} + x_{22} - y_2 \geqslant 0 \tag{4.60}$$
$$x_{31} + x_{32} - y_3 \geqslant 0 \tag{4.61}$$
$$x_{11} + x_{21} + y_4 \leqslant 2 \tag{4.62}$$
$$x_{11} + x_{32} + y_5 \leqslant 2 \tag{4.63}$$
$$x_{21} + x_{31} + y_6 \leqslant 2 \tag{4.64}$$
$$x_{12} + x_{22} + y_7 \leqslant 2 \tag{4.65}$$
$$x_{12} + x_{32} + y_8 \leqslant 2 \tag{4.66}$$
$$x_{22} + x_{32} + y_9 \leqslant 2 \tag{4.67}$$
$$x_{ij} \epsilon \{0, 1\}$$

The LP relaxation produces a fractional solution. Proceeding to an optimal IP solution the maximal objective value is 8, showing that the model is not satisfiable. Among the alternative optimal integer solutions is

$$x_{11} = x_{22} = x_{32} = 1, x_{12} = x_{21} = x_{31} = 0 \tag{4.68}$$

$$y_1 = y_2 = y_3 = y_4 = y_5 = y_6 = y_7 = y_8 = 1, y_9 = 0 \tag{4.69}$$

i.e. we can satisfy at most eight clauses.

(This has the 'useless' interpretation that we can put object 1 in box 1 and objects 2 and 3 in box 2 so satisfying the conditions that every object must be put somewhere but breaking one of the conditions that no more than one object can be put in any one box.) As a variant of the maximum satisfiability problem we might weight clauses differently according to their size. For example we might prefer smaller clauses to larger ones. This could easily be accomplished by giving different weights to the variables in the objective.

Although resolution and absorption produce the minimal size implied clauses their complete sum may contain redundancies, i.e. the conjunction of a *proper subset*

of the prime implications may be equivalent to the original statement. We illustrate this by an example.

4.6 Simplest Equivalent Logical Statement

Example 4.4 What is the minimum number of prime (disjunctive) clauses whose conjunction is equivalent to the following statement?

$$\cdot\, X_1 \vee X_2 \vee X_4 \tag{4.70}$$
$$\cdot\, X_1 \vee \overline{X_2} \vee X_4 \tag{4.71}$$
$$\cdot\, \overline{X_1} \vee \overline{X_4} \vee X_5 \tag{4.72}$$
$$\cdot\, \overline{X_1} \vee \overline{X_4} \vee \overline{X_5} \tag{4.73}$$
$$\cdot\, \overline{X_2} \vee X_3 \vee X_5 \tag{4.74}$$
$$\cdot\, \overline{X_2} \vee \overline{X_3} \vee X_5 \tag{4.75}$$
$$\cdot\, X_2 \vee X_3 \vee \overline{X_4} \tag{4.76}$$
$$\cdot\, X_1 \vee X_2 \vee X_3 \tag{4.77}$$
$$\cdot\, X_3 \vee X_4 \vee X_5 \tag{4.78}$$
$$\cdot\, X_1 \vee X_3 \vee X_5 \tag{4.79}$$

Applying resolution and absorption successively delivers the following set of prime implications, giving the following equivalent statement:

$$X_1 \vee X_4 \tag{4.80}$$
$$\cdot \overline{X_1} \vee \overline{X_4} \tag{4.81}$$
$$\cdot \overline{X_2} \vee X_5 \tag{4.82}$$
$$\cdot X_3 \vee X_5 \tag{4.83}$$
$$\cdot X_1 \vee X_2 \vee X_3 \tag{4.84}$$
$$\cdot X_2 \vee X_3 \vee \overline{X_4} \tag{4.85}$$

There may be a proper subset of these prime implications, the conjunction of which is also equivalent to the original statement. We give an IP formulation later. We can solve this example by means of a truth table with 32 rows. It is sufficient to find a subset of the clauses the truth of which implies the truth of all the clauses. (Implication the other way follows from the fact that the clauses are already prime *implications*.) Therefore we need to find a subset, the truth of which implies the truth of the other clauses.

It can be shown that the first five clauses imply the sixth. Also the first four and the sixth clause imply the fifth. No smaller subset implies the others. Therefore there are two shortest equivalent statements. They are

$$(X_1 \vee X_4) \cdot (\overline{X_1} \vee \overline{X_4}) \cdot (\overline{X_2} \vee X_5) \cdot (X_3 \vee X_5) \cdot (X_1 \vee X_2 \vee X_3) \tag{4.86}$$

$$(X_1 \vee X_4) \cdot (\overline{X_1} \vee \overline{X_4}) \cdot (\overline{X_2} \vee X_5) \cdot (X_3 \vee X_5) \cdot (X_2 \vee X_3 \vee \overline{X_4}) \tag{4.87}$$

In practice *heuristics* are often used to find, hopefully good, but suboptimal solutions to this problem. However, in this book, we concentrate on exact methods.

Figure 4.2 demonstrates the dependencies of the prime implications.

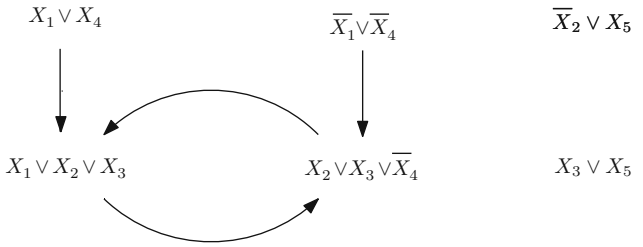


Fig. 4.2 Dependency between prime implications

If a clause has arrows leading into it then it is the result of successive resolutions applied to the clauses from where the arrows originate and the resultant clauses. It is therefore redundant when all the clauses at the origins of the arrows are present. Notice, therefore, that $X_1 \vee X_2 \vee X_3$ is redundant if $X_2 \vee X_3 \vee \overline{X_4}$ is present, being the resolvent of this clause and $X_1 \vee X_4$. However, $X_2 \vee X_3 \vee \overline{X_4}$ is the resolvent of $X_1 \vee X_2 \vee X_3$ and $\overline{X_1} \vee \overline{X_4}$. Therefore $X_1 \vee X_2 \vee X_3$ and $X_2 \vee X_3 \vee \overline{X_4}$ are each only redundant if the other is present.

We can use the dependency diagram to give an IP formulation of the problem of finding the minimum sum of prime implications. We introduce 0–1 variables y_i which are 1 if the corresponding clause is present. For each clause we keep track of which sets (if any) of prime implication clauses give rise to it by successive resolutions applied to them. Then this clause is redundant if any of the origin sets of clauses is present. Each origin set of clauses $\{i_1, i_2, \ldots, i_r\}$ is represented by a 0–1 variable $z_{i_1,i_2,\ldots,i_r}$ which takes the value 1 if all the clauses in the set are present.

Clause i may have a number of alternative origin sets $S_i^1, S_i^2, \ldots, S_i^M$. In order to avoid ignoring any origin set we must suspend the absorption operation when using resolution for this purpose. Clause i can only be removed if all the clauses in one of the origin sets is present. Therefore we stipulate

$$y_i + \sum_l z_{S_i^l} \geq 1 \quad \text{for all } i \tag{4.88}$$

If a clause has no origin sets then it must be present and there is no $\sum_l z_{S_i^l}$ term.

In order to ensure that if $z_{i_1,i_2,\ldots,i_r} = 1$ clauses $i_1, i_2, \ldots, i_r$ are present we have

$$z_{i_1,i_2,\ldots,i_r} - y_{i_j} \leq 0 \quad \text{for all } i_j \tag{4.89}$$

The objective

$$\text{Minimise} \quad \sum_i y_i \tag{4.90}$$

enables the minimum sum of prime implications to be found.

For our (simple) example we have the model

$$\begin{aligned}
\text{Minimise} \quad & y_1 + y_2 + y_3 + y_4 + y_5 + y_6 && (4.91)\\
& y_1,\ y_2,\ y_3,\ y_4 \geqslant 1 && (4.92)\\
& y_5 + z_{1,6} \geqslant 1 && (4.93)\\
& y_6 + z_{2,5} \geqslant 1 && (4.94)\\
& z_{1,6} - y_1 \leqslant 0 && (4.95)\\
& z_{1,6} - y_6 \leqslant 0 && (4.96)\\
& z_{2,5} - y_2 \leqslant 0 && (4.97)\\
& z_{2,5} - y_5 \leqslant 0 && (4.98)
\end{aligned}$$

We can simplify a statement in DNF to produce its *prime implicants* using the (logical) dual of resolution. A prime implicant is a conjunctive clause which *implies* the original statement and is such that no smaller clause containing a subset of the literals with the same sign implies the statement. The corresponding operation to resolution is known as *consensus.* A consensus of two (conjunctive) clauses with overlapping literals of the same sign, except for one literal, is obtained by taking their conjunction and deleting the common literal which is negated in one and unnegated in the other. For example the consensus of

$$X_1 \cdot X_2 \cdot X_3 \ \vee \ X_2 \cdot \overline{X_3} \cdot X_4 \tag{4.99}$$

is

$$X_1 \cdot X_2 \cdot X_4 \tag{4.100}$$

The consensus clearly *implies* the pair of clauses from which it is derived. It is added to the set of clauses.

We also apply *absorption*. If one conjunctive clause contains literals (of the same sign) which are a subset of the other then the larger clause is redundant and is removed. If we successively apply consensus and absorption to a statement in DNF we obtain the disjunction of all the prime implicants. This statement is equivalent to the original statement. Note the distinction between simplification in DNF and CNF. In CNF we obtain prime *implications* but in DNF we obtain prime *implicants.* But analogous to the CNF case, the disjunction of a proper subset of these prime

implicants may also be equivalent to the original statement. In order to illustrate this let us consider the negation of the statement in Example 4.4. Using De Morgan's laws (as described in Chapter 1) this is naturally written in DNF by interchanging '$\cdot$' and '$\vee$' and changing the sign of each literal. Applying consensus and absorption delivers us the logical dual of (3.80)–(3.85), i.e.

$$(\overline{X_1} \cdot \overline{X_4}) \vee (X_1 \cdot X_4) \vee (X_2 \cdot \overline{X_5}) \vee (\overline{X_3} \cdot \overline{X_5}) \vee (\overline{X_1} \cdot \overline{X_2} \cdot \overline{X_3}) \vee (\overline{X_2} \cdot \overline{X_3} \cdot X_4) \quad (4.101)$$

The minimum disjunction of prime implicants, giving an equivalent statement, consists of either the first five or the first four and the sixth of the above conjunctive clauses corresponding to the duals of (3.86) and (3.87).

We have confined our attention to minimising the number of clauses in an equivalent statement (whether in CNF or DNF). Alternatively we might wish to minimise the total number of literals in the equivalent statement. In order to do this we would weight clauses according to their size in the minimisation. For the example this still results in (4.86) and (4.87).

It will clearly never be the case that the statement using the minimum number of literals does not use prime implications (or prime implicants in the DNF case).

CNF and DNF are only two of the normal forms which we might wish to minimise. As discussed in Chapter 1 there are many ways to represent a logical statement as well as different connectives which can be used. In practical terms the relevant form to use depends on the problem being considered. For example in Sect. 4.10 we discuss different forms of logical circuit design where we might wish to minimise the number of switches or gates, or some other aspects of the architecture.

4.7 Horn Clauses: Simple Satisfiability Problems

The satisfiability problem and its extensions are very difficult problems to solve in the worst case (they are $\mathbb{NP}$ hard) whether they are solved by logical or IP methods.

However there are special cases that are comparatively easy to solve. One such is when all the clauses are *Horn clauses.* These are statements which can be written in the form

$$X_1 \cdot X_2 \cdot X_3 \longrightarrow X_4 \quad (4.102)$$

or

$$X_1 \cdot X_2 \cdot X_3 \longrightarrow \mathbf{T} \quad (4.103)$$

i.e. they can be written as implications where the premises are conjunctions of unnegated literals and the consequents are either single unnegated literals or tautologies (always true). Written as disjunctive clauses (4.102) and (4.103) are

$$\overline{X_1} \vee \overline{X_2} \vee \overline{X_3} \vee X_4 \tag{4.104}$$

and

$$\overline{X_1} \vee \overline{X_2} \vee \overline{X_3} \tag{4.105}$$

Hence Horn clauses, expressed as disjunctions, contain *at most* one unnegated literal. This makes such systems particularly easy to solve (the number of steps is always a polynomial function – which can be made a linear function – of the size of the problem). A number of logic programming systems restrict themselves to rules of this form for this reason. Some systems that are not Horn can be made such by a simple transformation of variables (e.g. replacing unnegated literals by negated ones, throughout the system, or vice versa). Such systems are called *essentially Horn*.

We illustrate the situation by means of the following example.

Example 4.5 From the following set of Horn clauses is it valid to deduce $\overline{X_2}$?

$$\overline{X_1} \vee \overline{X_2} \vee X_3 \tag{4.106}$$

$$\cdot\ X_1 \vee \overline{X_2} \tag{4.107}$$

$$\cdot\ \overline{X_2} \vee X_4 \tag{4.108}$$

$$\cdot\ \overline{X_1} \vee \overline{X_3} \vee \overline{X_4} \tag{4.109}$$

As explained, at the beginning of this chapter, we negate the conclusion, appending

$$\cdot\ X_2 \tag{4.110}$$

as an extra Horn clause and test the satisfiability of the system. If it is not satisfiable then the conclusion $\overline{X_2}$ is valid. (Alternatively we could deduce all the prime implications of the original system to see if $\overline{X_2}$ is one of them.)

If we apply resolution to the system we obtain the tree in Fig. 4.3.

Notice that one of the two clauses in each resolution operation is one of the input clauses. This is in contrast to Fig. 4.1 where the last resolution was between two non-input clauses. If each resolution involves an input clause it is only necessary to store them and the current resolved clause. This leads to a reduction in the amount of computation to a polynomial number of steps. (In fact it can be made a linear number of steps.) In order to see why *input resolution* is sufficient for delivering all the prime implications of a Horn clause system we consider the following (totally general) example of two successive resolutions illustrated in Fig. 4.4.

C_1, C_2, C_3, C_4 are disjunctive clauses and the input clauses are at the top level. In order for the system to be Horn and for the resolutions to be possible we stipulate

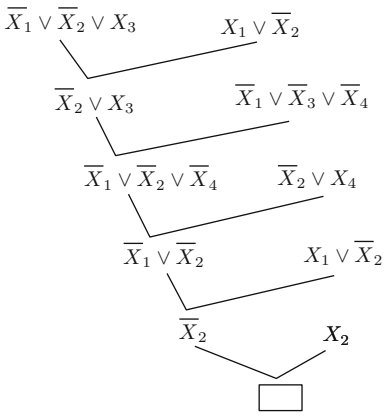


Fig. 4.3 A resolution tree for a Horn system

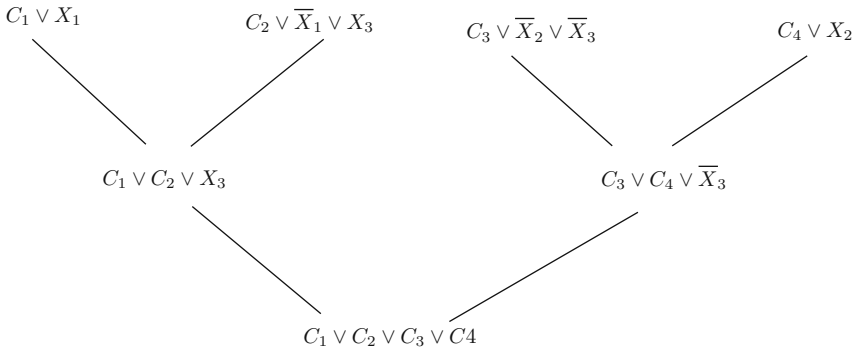

$C_1 \vee X_1$
$C_2 \vee \overline{X}_1 \vee X_3$
$C_3 \vee \overline{X}_2 \vee \overline{X}_3$
$C_4 \vee X_2$
$C_1 \vee C_2 \vee X_3$
$C_3 \vee C_4 \vee \overline{X}_3$
$C_1 \vee C_2 \vee C_3 \vee C4$

Fig. 4.4 Non-input resolution in a Horn system

that all the literals in C_1, C_2, C_3 and C_4 are negated. (Some literals may be shared in common.)

At the top level we eliminate X_1 and X_2, respectively, and at the next level we eliminate X_3.

Observe that Fig. 4.4 illustrates a *non-input* resolution since both clauses at the second level are non-input. But it is possible to derive the same result by the *input* resolution illustrated in Fig. 4.5.

Therefore we can restrict ourselves to input resolution when we have Horn systems. Another feature of Horn systems, worth remarking on, is that they are *closed* under resolution. Each resolved clause is itself Horn.

It also turns out that the corresponding IP model for a Horn system is particularly easy to solve. We illustrate the combining of constraints in such a model, corresponding to Example 4.5, in Fig. 4.6.

In order to simplify the presentation we do not explicitly include the constraints $x_i \geqslant 0$ and $-x_i \geqslant -1$ which are added, where necessary, to make coefficients equal.

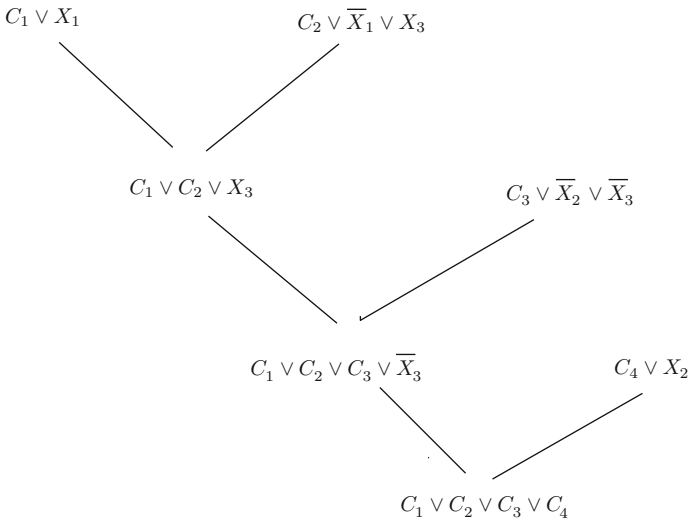

$C_1 \vee X_1$
$C_2 \vee \overline{X}_1 \vee X_3$
$C_1 \vee C_2 \vee X_3$
$C_3 \vee \overline{X}_2 \vee \overline{X}_3$
$C_1 \vee C_2 \vee C_3 \vee \overline{X}_3$
$C_4 \vee X_2$
$C_1 \vee C_2 \vee C_3 \vee C_4$

Fig. 4.5 Input resolution in a Horn system

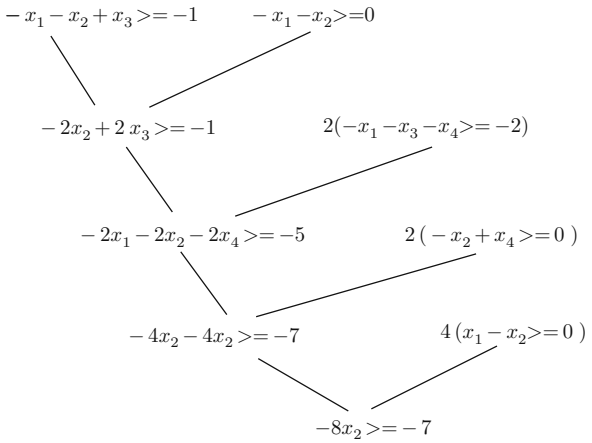


Fig. 4.6 Input resolution for an equation system

The final constraint, after rounding, clearly demonstrates that $x_2 = 0$.

Figure 4.6 also shows that, with input resolution applied to constraints, we can postpone rounding until the end. Consider a constraint representing a Horn clause.

$$-\sum_{i=1}^{n} x_i + x_{n+1} \geqslant 1 - n \tag{4.111}$$

(the variable x_{n+1} may or may not be present). After an application of resolution we still obtain a Horn clause which, with rounding, would produce a clause of the form (4.111). Therefore before rounding it would take the form

$$-\sum_{i=1}^{n} x_i + x_{n+1} \geqslant 1 - n - \varepsilon \tag{4.112}$$

where

$$0 \leqslant \varepsilon < 1$$

A subsequent resolution, involving (4.112), would also involve an input clause with an integer right-hand side. These clauses, when added together, and divided by a positive integer p (the left-hand side common coefficient) would result in a right-hand side of the form

$$N - \frac{\varepsilon}{p} \tag{4.113}$$

The result, after rounding, must again be a Horn clause of the form (4.111). Hence postponing rounding until the end still allows us to deduce a constraint representing the final conclusion.

It is therefore a feature of the IP formulation of Horn systems that they can be solved as LPs together with a final rounding to the optimal objective value. Since there exist polynomially bounded algorithms for LP this again demonstrates the computationally easier nature of Horn systems. The feasibility (satisfiability) of a system can therefore be tested by LP. A system will be satisfiable if and only if the LP relaxation of the corresponding IP model is feasible.

Here we are effectively applying (the LP form of) Fourier–Motzkin elimination as explained in Chapter 2.

4.8 Constraint Logic Programming

Constraint logic programming (CLP) is, in many ways, better called *constraint satisfaction* and uses relatively little 'logic'. It is an alternative approach to solving many of the problems discussed in this book. It is sometimes, but not always, a faster method. Although it is not based on the methods of LP and IP some of the operations (e.g. *branching)* used are similar to those of IP. Since it does not rely on the rich computational methodology of LP it lacks the mathematical sophistication of LP and IP. However, it is much richer in its modelling capabilities and more flexible in its solution strategies. These aspects are discussed in subsequent parts of this section. One of the reasons it proves powerful for some problems is the rather mundane one that computers have become very fast. Even complete enumeration

has become viable for some problems. CLP has many features in common with the *reduce* (presolve) procedures now commonly used to preprocess models.

CLP is not designed as an *optimisation method* although it can be adapted to such by making the objective, with progressively tighter bounds, a constraint.

So far there has been comparatively little success in combining the two approaches into 'hybrid' systems although there have been a number of designs for such systems. Some of these are discussed subsequently.

CLP can be regarded as a 'procedural' language as it is possible to model conditions, in the statement of the problem, which help to direct the computational search.

4.8.1 Modelling in CLP

LP and IP restrict the modelling to linear expressions of the form '$\leqslant$', '$\geqslant$' and '$=$'. In many situations it is convenient to model many more conditions. This is usually done in CLP in the form of predicates. Of course it is possible to convert these predicates into conventional IP constraints but the conversion is often cumbersome. We discuss some of the most widely used predicates below. These are usually referred to as *global constraints* as they are in-built into the computer software used. In contrast it is also usually possible to define *local constraints* specific to the problem being modelled. These constraints may serve the purpose of modelling specific conditions. They may also serve the purpose of directing or restricting the search for solutions in a way which will speed the computation. For example constraints that rule out symmetrically equivalent solutions are often highly effective.

We describe some of the most commonly used predicates below (which may take different names in different software systems). There are many more which can be found in the manuals of the relevant software systems. In all cases we are assuming the variables are restricted to a finite number of discrete values.

4.8.1.1 The $\neq$ Constraint

This may take the form

$$\sum_j a_j x_j \neq b \tag{4.114}$$

in contrast to LP and IP constraints. Alternatively it may take the form

$$x \neq \{v_1, v_2, \ldots, v_n\} \tag{4.115}$$

so preventing x taking one of the specified values. We could model (4.114) using IP (see Section 3.1) as

$$\sum_{jj} a_j x_j - (M - b + \varepsilon)\delta \leq b - \varepsilon \tag{4.116}$$

$$\sum_{j} a_j x_j + (m - b - \varepsilon)\delta \geqslant m \tag{4.117}$$

where M and m are, respectively, known upper and lower bounds on $\sum_j a_j x_j$ and ε is a 'tolerance' above or below which the quantity is not regarded as equal to b.

We emphasise, however, that (4.114) is an alternative, less cumbersome, way of modelling the condition.

Ways of modelling some of the other predicates below are left as exercises in Sect. 4.12.

4.8.1.2 The 'All-Different' Constraint

This condition frequently arises in practical problems and is conveniently written as

$$all\_diff(x_1, x_2, \ldots, x_n) \tag{4.118}$$

It means that all the x_i must take *different* values.

For example *assignment* conditions that arise as part of many problems (apart from the 'pure' form of the 'easy' classical *assignment problem* mentioned in Chapter 2) can be modelled by (4.118). The value of x_i is interpreted as the name (number) of the position to which i is assigned and (4.118) prevents more than one i being assigned to any position.

4.8.1.3 The Cardinality Constraint

This condition can be written as

$$card_m(x_1, x_2, \ldots, x_n | v) \tag{4.119}$$

It means that exactly m of variables $x_1, x_2, \ldots, x_n$ must take the value v.

4.8.1.4 The 'At-Least' Constraint

This can be written as

$$at\_least_m(x_1, x_2, \ldots, x_n | v) \tag{4.120}$$

It means that at least m of variables $x_1, x_2, \ldots, x_n$ take the value v.

It was used as a modelling predicate in Sect. 3.5 in the form '*greater-than-or-equal' ('ge')* where the variables were Boolean and 'v' represented '**True'.** There it was allowed to take a nested form.

4.8.1.5 The 'Circuit' Constraint

This can be written as

$$circuit(x_1, x_2, \ldots, x_n) \tag{4.121}$$

where $(x_1, x_2, \ldots, x_n)$ is a permutation of integers $1, 2, \ldots, n$. x_i is the number of the integer in the circuit after i. For example *circuit*(5, 4, 1, 3, 2) represents the circuit shown in Fig. 4.7.

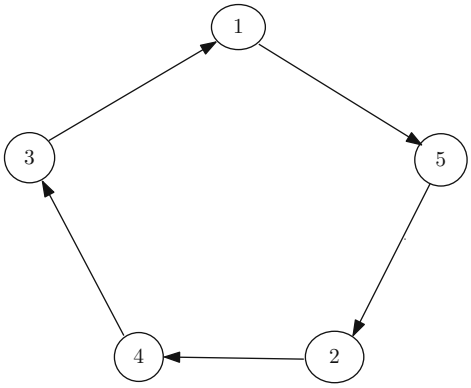


Fig. 4.7 A circuit

But it can easily be checked that the instantiated predicate *circuit*(6, 4, 7, 1, 3, 8, 5, 2) is false, i.e. the given permutation of numbers makes the predicate false as it does not represent a circuit (these are the 'subtours' illustrated in Fig. 2.16).

This predicate gives, in particular, a very compact formulation of the *travelling salesman problem* (TSP) (although the TSP is not efficiently solved by CLP).

4.8.2 Solving CLP Models

In the form of CLP which we are describing each *variable* in a model will have a finite *domain* of discrete values. These may be numbers or names, e.g.

$$x \in \{2, 5, 6, -1, 0\} \tag{4.122}$$

or

$$town \in \{Lewes, Edinburgh, Truro\} \tag{4.123}$$

Constraints (possibly in the form of predicates) will restrict the values which sets of variables can together take, e.g. suppose

$$x_1, x_2, x_3 \in \{1, 2, 3\} \tag{4.124}$$

$$all\_diff(x_1, x_2, x_3) \tag{4.125}$$

$$2x_1 + 3x_2 + x_3 \leqslant 10 \tag{4.126}$$

Since (4.126) applies we can restrict the domains of some of the variables to subsets of $\{1, 2, 3\}$.

Since

$$2x_1 \leqslant 10 - 3x_1 - x_3 \leqslant 10 - 5 = 5 \tag{4.127}$$

we must have $x_1 \leqslant 2$ restricting its domain to $\{1, 2\}$.

Similarly since

$$3x_2 \leqslant 10 - 2x_1 - x_3 \leqslant 6 \tag{4.128}$$

we have $x_2 \leqslant 2$ restricting its domain to $\{1, 2\}$.

Therefore, since x_1 and x_2 empty the domain $\{1, 2\}$, x_3 must take the value 3.

This implies

$$2x_1 + 3x_2 \leqslant 7 \tag{4.129}$$

Hence

$$3x_2 \leqslant 7 - 2x_1 \leqslant 5 \tag{4.130}$$

Therefore $x_2 \leqslant 1$ forcing $x_2 = 1$ and $x_1 = 2$.

The above type of argument is known as *constraint* propagation, i.e. using constraints, in conjunction with others, to restrict the domains of variables.

As mentioned earlier such an approach is very similar to the *presolve* procedure found in many traditional IP systems to 'simplify' models before solving them by more sophisticated LP/IP methods. In CLP (and IP) this can be extended to a *dynamic presolve* where, after setting certain variables to trial values, in a tree search one applies the approach to restrict the domains of subproblems down particular branches of the search tree.

After applying constraint propagation to restrict domains one is generally forced to try setting variables to values within their domains using a branching procedure. Then the implications of different trial values will be found, often demonstrating the impossibility of certain solutions (empty domains for some variables).

4.8.3 Hybrid CLP and IP systems

While CLP offers flexibility in modelling and search strategies IP relies on the strength of the LP (and possibly other) relaxations. There have been a number of

proposals to combine the two approaches into *'hybrid'* systems, most of which have not been fully realised. We outline possible ways of doing this.

4.8.3.1 Modelling in CLP to Produce IP Constraints

This approach uses the rich modelling capabilities of CLP to state the problem. The predicates (constraints) are then converted into traditional IP constraints. One powerful, universal, way of doing this using the 'ge' predicate was described in Sect. 3.5.

4.8.3.2 Transference of Information Between Two Systems

Here the idea is to solve a model using CLP and IP in parallel with exchange of information between the systems, e.g. CLP would provide restrictions on domains (akin to a dynamic presolve) while IP would provide increasingly tighter bounds on the objective by means of the LP relaxation (which could, itself, be used for further domain restriction).

If a model has a natural decomposible structure CLP can be used to solve subproblems whose solutions are incorporated into a *master* problem which is solved by IP.

There has been some success with this approach in producing special purpose methods for particular types of application. For example *column generation* is a method of solving important scheduling problems (particularly *crew scheduling* in the airline industry). Here columns ('rosters') are generated in the course of optimising a large 0–1 set-covering IP. Rather than generate the, potentially astronomic number of, columns all at once they are usually added in the course of optimisation. Their generation may effectively be done using CLP. The *cutting stock* problem also yields a model amenable to column generation. Here patterns for cutting up rolls of material can be generated in the course of optimisation (they are knapsack problems) and solved using CLP. The patterns give rise to columns of the 'master problem' to be solved by LP or IP. Another application is to maximising the throughput in a telecommunications network. Here different messages are routed through a network using different frequencies. There is a limit to the capacity of any one path. Different combinations of frequencies can be incorporated, within a capacity limit, along any path. Possible combinations can be found by CLP. The overall (discrete) multicommodity network flow model is then solved by IP.

4.8.3.3 Benders Decomposition

This is a powerful method of solving (usually mixed IP) structured models. For mixed IPs the method fixes the integer variables at trial values then solves the resultant LP. The result provides a *bound* on the optimal objective value of the original model. This is then incorporated into an IP model for finding the best values of the integer variables as an extra constraint (a 'Benders cut'). This IP is sometimes efficiently solved by CLP. Again the approach is usually problem specific.

4.9 Solving Integer Programmes as Satisfiability Problems

Most practical IP problems give rise to MIPs and are most efficiently solved using standard IP algorithms based on the LP relaxation and branch-and-bound augmented by cutting planes. However, the use of logical methods is very valuable in the formulation of these models. Badly built models may be prohibitively difficult to solve (even if they are accurately built). Logic provides the rigour and systematisation necessary.

PIP models arise less frequently although there are a number of classic applications. Usually the variables are restricted to be 0–1 although, so long as they are bounded and restricted to small integers, they can be converted to 0–1 variables by the method described in Chapter 2.

The 0–1 PIP models can always be converted into satisfiability problems, and sometimes be solved efficiently as such by, e.g. the Davis–Putnam–Loveland procedure. Even if they are not solved in this way their conversion into a logical form produces cutting planes which may be incorporated into the conventional IP methods.

In order to illustrate the method we consider a small 0–1 PIP example.

Example 4.6 Convert the following model into a satisfiability problem and solve it by the Davis–Putnam–Loveland (DPL) procedure:

$$\begin{aligned}
\text{Maximise} \quad & 21x_1 + 32x_2 + 45x_3 + 15x_4 + 19x_5 + 42x_6 + 15x_7 \\
& + 37x_8 + 50x_9 + 47x_{10} && (4.131)\\
\text{subject to} \quad & 20x_1 + 34x_2 + 42x_3 + 11x_4 + 48x_5 + 39x_6 + 53x_7 + 23x_8 + 46x_9 \\
& + 38x_{10} \leqslant 63 && (4.132)\\
& 19x_1 + 35x_2 + 23x_3 + 23x_4 + 39x_5 + 35x_6 + 45x_7 + 61x_8 + 47x_9 \\
& + 33x_{10} \leqslant 72 && (4.133)
\end{aligned}$$

$$x_1, x_2, \ldots, x_{10} \in \{0, 1\}$$

We represent the setting of each 0–1 variable x_i to 1 or 0 by the truth or falsity of an atomic statement X_i. Each of the constraints imposes restrictions on the possible, combined, values of the x_i, which may be stated in compound statements involving the X_i. For example (4.132) demonstrates that both x_2 and x_3 cannot be 1 since the sum of their coefficients in (4.132) exceeds the right-hand side. This can be represented by the statement $\overline{X}_2 \vee \overline{X}_3$.

In order to, systematically, find all such clauses implied by (4.132) it is convenient to place all the variables in ascending order of their coefficients as follows:

$$11x_4 + 20x_1 + 23x_8 + 34x_2 + 38x_{10} + 39x_6 + 42x_3 + 46x_9 + 48x_5 + 53x_7 \leqslant 63 \quad (4.134)$$

Working from the left we can see if

i. There are any single statements which must be false (i.e. whose coefficients exceed the right-hand side). In this example there are none.
ii. There are any pairs which cannot simultaneously be true. This gives the following conjunction of disjunctions:

$$
\begin{aligned}
&\overline{X}_4 \vee \overline{X}_7 \\
&\cdot\overline{X}_1 \vee \overline{X}_9 \\
&\cdot\overline{X}_1 \vee \overline{X}_5 \\
&\cdot\overline{X}_1 \vee \overline{X}_7 \\
&\cdot\overline{X}_8 \vee \overline{X}_3 \\
&\cdot\overline{X}_8 \vee \overline{X}_9 \\
&\cdot\overline{X}_8 \vee \overline{X}_5 \\
&\cdot\overline{X}_8 \vee \overline{X}_7 \\
&\cdot\overline{X}_4 \vee \overline{X}_9 \\
&\cdot\overline{X}_2 \vee \overline{X}_{10} \\
&\cdot\overline{X}_2 \vee \overline{X}_6 \\
&\cdot\overline{X}_2 \vee \overline{X}_3 \\
&\cdot\overline{X}_2 \vee \overline{X}_9 \\
&\cdot\overline{X}_2 \vee \overline{X}_5 \\
&\cdot\overline{X}_2 \vee \overline{X}_7 \\
&\cdot\overline{X}_{10} \vee \overline{X}_6 \\
&\cdot\overline{X}_{10} \vee \overline{X}_3 \\
&\cdot\overline{X}_{10} \vee \overline{X}_9 \\
&\cdot\overline{X}_{10} \vee \overline{X}_5 \\
&\cdot\overline{X}_{10} \vee \overline{X}_7 \\
&\cdot\overline{X}_6 \vee \overline{X}_3 \\
&\cdot\overline{X}_6 \vee \overline{X}_9 \\
&\cdot\overline{X}_6 \vee \overline{X}_5 \\
&\cdot\overline{X}_6 \vee \overline{X}_7 \\
&\cdot\overline{X}_3 \vee \overline{X}_9 \\
&\cdot\overline{X}_3 \vee \overline{X}_5 \\
&\cdot\overline{X}_3 \vee \overline{X}_7 \\
&\cdot\overline{X}_9 \vee \overline{X}_5 \\
&\cdot\overline{X}_9 \vee \overline{X}_7 \\
&\cdot\overline{X}_5 \vee \overline{X}_7
\end{aligned}
\tag{4.135}
$$

We could now proceed to enumerate all triples, the sum of whose coefficients exceeds the right-hand side, and give the corresponding logical statement and then

all subsets of cardinality 4 and so on. Of course it would be redundant to enumerate sets whose members contain all those already represented by a subset of the set. The resulting disjunction would be absorbed by the smaller disjunction. If we were to proceed in this manner we would eventually capture the full import of constraint (4.132) in a conjunction of disjunctive clauses. However, to avoid the potentially explosive build up in clauses we will not proceed beyond pairs of cardinality 2 at this stage.

Carrying out a similar procedure for constraint (4.133) produces some of the disjunctions in (4.135) together with

$$\begin{aligned} &\overline{X}_2 \vee \overline{X}_8 \\ &\cdot\overline{X}_6 \vee \overline{X}_8 \\ &\cdot\overline{X}_8 \vee \overline{X}_{10} \\ &\cdot\overline{X}_9 \vee \overline{X}_{10} \\ &\cdot\overline{X}_1 \vee \overline{X}_8 \\ &\cdot\overline{X}_4 \vee \overline{X}_8 \end{aligned} \tag{4.136}$$

Again we will, temporarily, content ourselves with disjunctions of pairs of literals.

In order to deal with the objective we place constraints on its value and formulate the resulting logical implication.

In this example the maximum possible value of the objective function is 323 (the sum of the objective coefficients) and the minimum value is 0. An efficient way of proceeding is to do a 'binary search' and progressively dissect the range of values in which the optimal objective value might lie. Therefore we begin by seeing if half the maximum objective value is obtainable. This is 161 (with rounding). Therefore we impose the trial constraint

$$21x_1 32x_2 + 45x_3 + 15x_4 + 19x_5 + 42x_6 + 15x_7 + 37x_8 + 50x_9 + 47x_{10} \geqslant 161 \tag{4.137}$$

In terms of complemented variables ($\overline{x}_i = 1 - x_i$) this may be written as

$$21\overline{x}_1 + 32\overline{x}_2 + 45\overline{x}_3 + 15\overline{x}_4 + 19\overline{x}_5 + 42\overline{x}_6 + 15\overline{x}_7 + 37\overline{x}_8 + 50\overline{x}_9 + 47\overline{x}_{10} \leqslant 162 \tag{4.138}$$

We treat this constraint, as with the others, and express it in a logical form.

The smallest subsets of these (complemented) variables, which give rise to logical conjunctions, are of cardinality 4 giving

$$
\begin{aligned}
& X_1 \vee X_3 \vee X_9 \vee X_{10} \\
& \cdot X_2 \vee X_3 \vee X_8 \vee X_9 \\
& \cdot X_2 \vee X_8 \vee X_9 \vee X_{10} \\
& \cdot X_2 \vee X_3 \vee X_6 \vee X_{10} \\
& \cdot X_2 \vee X_3 \vee X_6 \vee X_9 \\
& \cdot X_2 \vee X_6 \vee X_9 \vee X_{10} \\
& \cdot X_2 \vee X_3 \vee X_9 \vee X_{10} \\
& \cdot X_3 \vee X_6 \vee X_8 \vee X_{10} \\
& \cdot X_3 \vee X_6 \vee X_8 \vee X_9 \\
& \cdot X_6 \vee X_8 \vee X_9 \vee X_{10} \\
& \cdot X_3 \vee X_8 \vee X_9 \vee X_{10} \\
& \cdot X_3 \vee X_6 \vee X_9 \vee X_{10}
\end{aligned}
\tag{4.139}
$$

Again we will not proceed, at this stage, with enumerating the larger clauses.

We take the conjunction of (4.135), (4.136) and (4.139) and see if it is satisfiable. Since resolution (together with absorption) is a 'complete refutation procedure' it is possible to determine if a statement, such as this, is unsatisfiable without resorting to the branching in the DPL procedure.

We resolve the fourth disjunction in (4.139), and its successive resolvents, with each of $\overline{X}_6 \vee \overline{X}_9$, $\overline{X}_3 \vee \overline{X}_9$, $\overline{X}_2 \vee \overline{X}_9$, $\overline{X}_{10} \vee \overline{X}_9$ in turn to produce $\overline{X_9}$. Similarly we resolve the 10th disjunction of (4.139) with each of $\overline{X}_6 \vee \overline{X}_3$, $\overline{X}_{10} \vee \overline{X}_3$, $\overline{X}_3 \vee \overline{X}_9$, $\overline{X}_8 \vee \overline{X}_3$ in turn to produce $\overline{X_3}$. Resolving the fifth disjunction of (4.139) with $\overline{X}_2 \vee \overline{X}_8$, $\overline{X}_6 \vee \overline{X}_8$, $\overline{X}_6 \vee \overline{X}_3$, $\overline{X}_8 \vee \overline{X}_9$ in turn produces $\overline{X_8}$. Resolving the eighth disjunction of (4.139) with $\overline{X_2} \vee \overline{X}_{10}$, $\overline{X}_2 \vee \overline{X}_8$, $\overline{X}_2 \vee \overline{X}_6$, $\overline{X}_2 \vee \overline{X}_3$ in turn produces $\overline{X}_2$. These four resolvents resolve with the second disjunction in (4.139) to produce an empty clause (a contradiction), showing the objective of 161 is unobtainable.

Note that we have managed to show that a 'relaxation' (only using some of the implied disjunctive clauses) is unsatisfiable and that, therefore, the full logical representation is unsatisfiable, i.e. it was unnecessary to generate all the implied disjunctive clauses.We could have sufficed with even fewer clauses if we had a judicious way of choosing them.

We therefore further dissect the possible objective values from 0 to 80 by imposing the constraint

$$
21\overline{x}_1 + 32\overline{x}_2 + 45\overline{x}_3 + 15\overline{x}_4 + 19\overline{x}_5 + 42\overline{x}_6 + 15\overline{x}_7 + 37\overline{x}_8 + 50\overline{x}_9 + 47\overline{x}_{10} \leqslant 243 \tag{4.140}
$$

From this the smallest clause we can create is

$$
X_2 \vee X_8 \vee X_6 \vee X_3 \vee X_{10} \vee X_9 \tag{4.141}
$$

Appending this to (4.135) and (4.136) and applying the DLP procedure (Exercise 4.12.24) shows the compound statement to be satisfiable. A satisfying

set of truth values corresponds to the solution is

$$x_1 = x_4 = x_{10} = 1, \quad x_2 = x_3 = x_5 = x_6 = x_7 = x_8 = x_9 = 0 \tag{4.142}$$

However, since the logical statement is a relaxation of the full problem we must check if this solution satisfies the constraints (4.132), (4.133) and (4.140). It clearly breaks them all. Therefore we must add further disjunctions to rule out this solution. We begin by appending only

$$\overline{X}_1 \vee \overline{X}_4 \vee \overline{X}_{10} \tag{4.143}$$

The DPL procedure again shows this to be satisfiable with a satisfying set of truth values

$$x_1 = x_3 = x_4 = 1, \quad x_2 = x_5 = x_6 = x_7 = x_8 = x_9 = x_{10} = 0 \tag{4.144}$$

But it can be seen that this solution breaks (4.134). We therefore append the disjunction

$$\overline{X}_1 \vee \overline{X}_3 \vee \overline{X}_4 \tag{4.145}$$

Applying resolution demonstrates that the full statement is not satisfiable, showing that an objective value of 80 is not attainable.

We therefore further dissect the possible objective values to lie between 0 and 40 by logically representing

$$21\overline{x}_1 + 32\overline{x}_2 + 45\overline{x}_3 + 15\overline{x}_4 + 19\overline{x}_5 + 42\overline{x}_6 + 15\overline{x}_7 + 37\overline{x}_8 + 50\overline{x}_9 + 47\overline{x}_{10} \leqslant 283 \tag{4.146}$$

The smallest clauses implied by this constraint are

$$\begin{aligned} &X_1 \vee X_2 \vee X_3 \vee X_6 \vee X_7 \vee X_8 \vee X_9 \vee X_{10} \\ &X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_6 \vee X_8 \vee X_9 \vee X_{10} \\ &X_2 \vee X_3 \vee X_5 \vee X_6 \vee X_7 \vee X_8 \vee X_9 \vee X_{10} \\ &X_2 \vee X_3 \vee X_4 \vee X_5 \vee X_6 \vee X_8 \vee X_9 \vee X_{10} \end{aligned} \tag{4.147}$$

Appending (4.147) to (4.135) and (4.136) and applying the DPL method shows the resulting statement to be satisfiable with the two possible solutions

$$x_1 = x_6 = 1, \quad x_2 = x_3 = x_4 = x_5 = x_7 = x_8 = x_9 = x_{10} = 0 \tag{4.148}$$

and

$$x_1 = x_{10} = 1, \quad x_2 = x_3 = x_4 = x_5 = x_6 = x_7 = x_8 = x_9 = 0 \tag{4.149}$$

Both these solutions satisfy the constraints of the original IP model.

The first solution results in an objective value of 63 and the second in an objective value of 68 showing solution (4.149) to be optimal.

If we had only produced one of the satisfying solutions (and not checked if there were others) we could have proceeded by dissecting the interval between the corresponding objective value and 80 progressively seeking, or showing the absence of solutions with larger objective values.

The full execution of use of resolution and the DPL procedure in this example is left as Exercise 4.12.24.

An alternative approach to examples such as this would be to use the LP relaxation to obtain upper bounds on the optimal objective value at each stage. These bounds could then be used as the right-hand side on the (objective) constraint. Also IP constraints could then be produced as 'cutting planes' from some of the resultant logical clauses in the manner described in Sect. 4.3. The application of this procedure to Example 4.6 is left as Exercise 4.12.26.

4.10 Applications

In this section we describe some practical problems that are particularly suited to being viewed as satisfiability problems, or extensions of this problem. They are often most satisfactorily solved using the methods of IP.

4.10.1 Electrical Circuit Design Using Switches

Suppose we wish to allow electric current to pass through a set of wires only if certain combinations of switches are on. There will be many ways of achieving this. Normally we will seek an 'economical' solution. Possible objectives are as follows:

i. Minimising the number of switches.
ii. Minimising the wire length.
iii. Minimising the number of junctions.

For example, Fig. 4.8 illustrates a circuit, with redundancies, which represents the logical statement (in DNF):

$$X_1 \cdot X_2 \vee X_1 \cdot \overline{X_2} \vee \overline{X_2} \cdot X_3 \tag{4.150}$$

Simplifying (4.150) (by consensus and absorption) we obtain

$$X_1 \vee \overline{X_2} \cdot X_3 \tag{4.151}$$

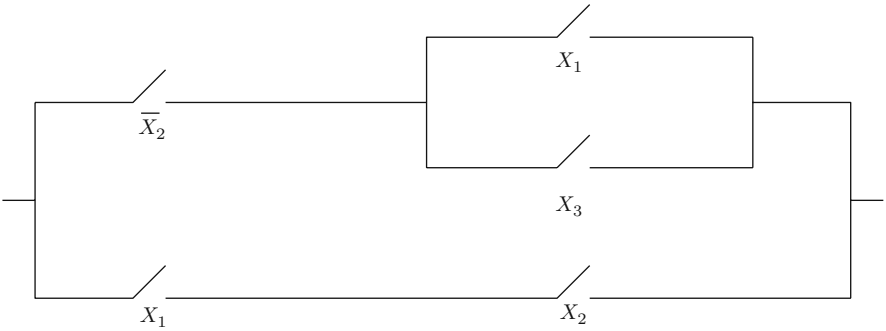


Fig. 4.8 A redundant electrical circuit

These two prime implicants represent the smallest statement in terms of both the number of clauses and number of literals. The result translates into the circuit in Fig. 4.9.

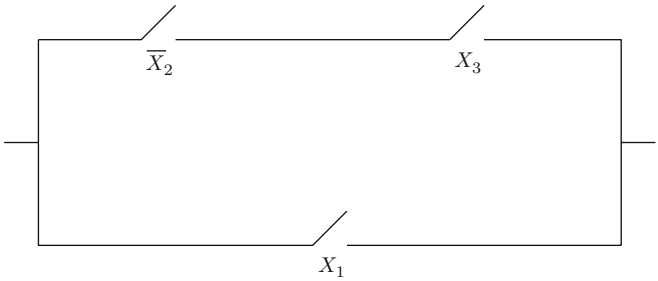


Fig. 4.9 A simplified electrical circuit

For more complicated circuits we would need to specify the exact objective and use the methods described in Section 4.5.

4.10.2 Logical Net Design Using Gates

Electronic components exist to perform different logic functions. In particular if one has components which perform one of the *complete connectives,* discussed in Chapter 1, then one can construct any logic function out of them by connecting together a number of these components. For example a common gate used performs the **NOR** function (the *connective arrow* '↓'). It is shown in Fig. 4.10.

This gate gives a signal Y when, and only when, *neither* input X_1 nor input X_2 is present, i.e. the logical function $\overline{X_1} \cdot \overline{X_2}$. By connecting together such gates one can perform any logical function. For example the logical statement

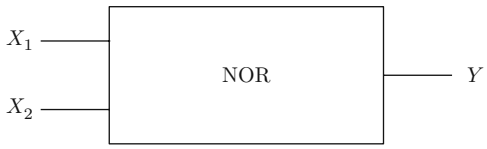


Fig. 4.10 A NOR gate

$$(X_1 \vee X_2) \cdot (\overline{X_1} \vee \overline{X_2}) \tag{4.152}$$

(known as 'exclusive or') is realised by the net in Fig. 4.11.

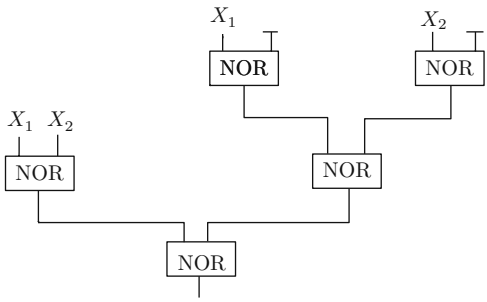


Fig. 4.11 The exclusive OR function

The minimum DNF or CNF realisation of (4.152) is of little use in designing a structure with the minimum number of gates to perform the function. Another 'normal form' is needed. However, the problem can be formulated as an IP and is set as Exercise 4.12.20. In fact Fig. 4.11 is the minimal representation.

It is worth pointing out that gates, such as **NOR,** can be connected to each other to give control systems with a 'memory'. For example Fig. 4.12 represents a 'flip-flop'.

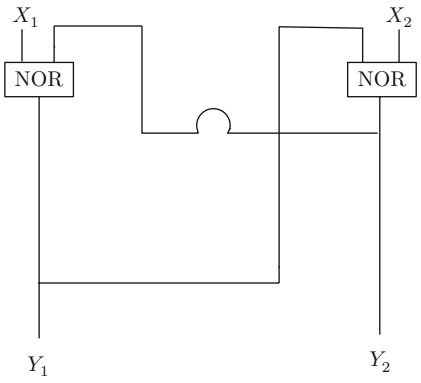


Fig. 4.12 A flip-flop

If the X_1 signal comes before X_2 then the first **NOR** gate is switched off. The only output will be Y_2. On the other hand if X_2 precedes X_1 the only output will be Y_1. The logic of optimally designing systems with memory is beyond the scope of this book but can be achieved by IP.

Many other gates are manufactured in practice performing functions such as **NAND, XOR** (exclusive or), **OR,** etc.

A particularly important class of gates is known as *'threshold gates'*. These have a number of inputs (n) which multiply the 0–1 input signals by given amounts $a_1, a_2, \ldots, a_n$ and only produce an output if the combined input reaches some designated threshold level b or more. A threshold gate is illustrated in Fig. 4.13.

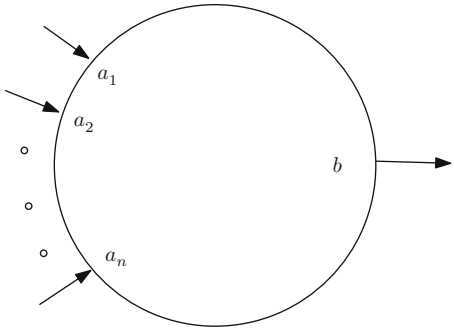


Fig. 4.13 A threshold gate

If the inputs are represented by 0–1 variables $x_1, x_2, \ldots, x_n$ then the gate produces only an output if the *knapsack* constraint

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b \tag{4.153}$$

is satisfied.

It is possible to create all the logic functions by individual or combinations of threshold functions. Exercise 4.12.22 addresses this question but further discussion is beyond the scope of this book.

All the design problems discussed here are usually considered under the heading of VLSI (very large-scale integrated circuit) design.

4.10.3 The Logical Analysis of Data (LAD)

A major application of computational logic is to finding 'economical' (or minimum size) logical formulae which can be used to predict a result from certain input data. Important areas are disease diagnosis, credit scoring, pattern recognition, data compression in order to transmit or store digital data and the design of safety systems. The data are converted into a finite set of true/false observations. If necessary continuous measurements are converted into 0–1 statements, e.g.

$$Y = \text{True if and only if} \quad a \leqslant y \leqslant b \tag{4.154}$$

We motivate the discussion by a medical example.

Example 4.7 Create a minimal logical formula to diagnose the occurrence of a disease as a result of the following observations:

'T' and 'F' denote positive and negative symptoms, respectively, and '–' indicates irrelevance of the symptom for the particular test.

In order to obtain a logical formula for predicting the disease we first express the presence of a symptom S_i by a literal X_i.

Table 4.2 is represented in DNF as follows:

$$X_1 \cdot X_2 \cdot X_4 \vee \overline{X_1} \cdot X_2 \cdot \overline{X_4} \cdot X_5 \vee \overline{X_1} \cdot X_2 \cdot X_3 \cdot X_4 \vee X_1 \cdot X_3 \cdot \overline{X_4} \tag{4.155}$$

Table 4.2 Disease Present

Symptoms	S_1	S_2	S_3	S_4	S_5
Test 1	**T**	**T**	–	**T**	–
Test 2	**F**	**T**	–	**F**	**T**
Test 3	**F**	**T**	**T**	**T**	–
Test 4	–	**T**	**T**	**F**	–

Applying consensus and absorption we obtain the complete disjunction of prime implicants as

$$X_1 \cdot X_2 \cdot X_4 \vee \overline{X_1} \cdot X_2 \cdot \overline{X_4} \cdot X_5 \vee X_2 \cdot X_3 \tag{4.156}$$

By means of a truth table or IP we can show that none of the prime implicants is redundant. Therefore (4.156) represents the minimum equivalent statement to Table 4.2 in DNF. However, we also need to ensure that Table 4.3 is consistent with Table 4.2.

Table 4.3 Disease Not Present

Symptoms	S_1	S_2	S_3	S_4	S_5
Test 5	**T**	**T**	**T**	**F**	–
Test 6	**T**	**F**	**T**	–	**F**
Test 7	**F**	**F**	–	–	–

Table 4.3 is represented by

$$X_1 \cdot X_2 \cdot X_3 \cdot X_4 \ \vee X_1 \cdot \overline{X}_2 \cdot X_3 \cdot \overline{X_5} \vee \overline{X_1} \cdot \overline{X_2} \tag{4.157}$$

Since this statement needs to be false, if the patient has the disease, we negate it using De Morgan's laws to give (in CNF)

$$(\overline{X_1} \vee \overline{X_2} \vee X_3 \vee X_4) \cdot \ (\overline{X_1} \vee X_2 \vee \overline{X}_3 \vee X_5) \cdot \ (X_1 \vee X_2) \tag{4.158}$$

If the patient has the disease this statement must be true to be consistent with (4.156).

Equation (4.158) can be simplified to

$$(\overline{X_1} \vee \overline{X_2} \vee X_3 \vee X_4) \cdot (X_2 \vee \overline{X}_3 \vee X_5) \cdot (X_1 \vee X_2) \tag{4.159}$$

Expressed in DNF this is (after simplification and taking out redundant prime implicants)

$$\overline{X_1} \cdot X_2 \vee X_2 \cdot X_3 \vee X_2 \cdot X_4 \vee X_1 \cdot \overline{X_2} \cdot \overline{X}_3 \vee X_1 \cdot \overline{X_2} \cdot X_5 \tag{4.160}$$

We must now check that the conjunction of (4.160) with (4.156) is consistent (satisfiable). This is left as Exercise 4.12.23.

Once it is shown to be satisfiable we can use (4.156) to define a new (smaller) set of tests for diagnosing the disease. This is given in Table 4.4.

Table 4.4 New Tests for the Disease

Symptoms	S_1	S_2	S_3	S_4	S_5
Test 1	**T**	**T**	–	**T**	–
Test 2	**F**	**T**	–	**F**	**T**
Test 3	**T**	**–**	**T**	**–**	–

As mentioned in Sect. 4.6 the finding of the minimum logical expression can be very difficult for larger examples and heuristics often have to be resorted to in order to find suboptimal solutions. However, the finding of the *minimum* expression is highly desirable in many of these problems in view of the cost or utility of the resulting test regime (in the case of LAD).

4.10.4 Chemical-processing networks

In this situation it is sometimes desirable to represent and analyse the logical dependence of different processes before solving the associated problem by IP. We illustrate this by the very small example in Fig. 4.14.

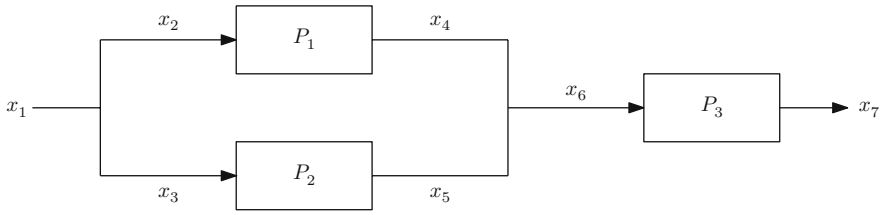


Fig. 4.14 A chemical-processing network

P1, P2 and P3 are chemical processes into and out of which are flows of material whose quantities are represented by the corresponding x_i variables. The processes have maximum capacities of inputs of M_1, M_2 and M_3, respectively. Their outputs are their inputs multiplied by α_1, α_2 and α_3, respectively. It is not possible to operate P1 and P2 simultaneously. If we represent whether each Pi is operating or not by 0–1 variables y_i then we have the following constraints:

$$x_1 - x_2 - x_3 = 0, \quad x_6 - x_4 - x_5 = 0 \tag{4.161}$$

$$x_4 - \alpha_1 x_2 = 0, \quad x_5 - \alpha_2 x_3 = 0, \quad x_7 - \alpha_3 x_6 = 0 \tag{4.162}$$

$$x_2 - M_1 y_1 \leq 0, \quad x_3 - M_2 y_2 \leq 0, \quad x_6 - M_3 y_3 \leq 0 \tag{4.163}$$

$$y_1 + y_2 \leq 1 \tag{4.164}$$

In addition we will have costs on x_1, x_3 and x_5 and a revenue on x_7 as well as (fixed) costs on y_1, y_2 and y_3.

As explained in Chapter 2 the standard version of the branch-and-bound algorithm relies on *dichotemies* based on branching on 0–1 variables. However in this type of application there is merit in analysing the logical relationships between the processes and representing them in a 'simple' form allowing more useful dichotemies. We do this using logical notation. Also it is worth representing, explicitly, the obvious condition (which would be satisfied at optimality) that if process P3 is not used then it is not worth using P1 or P2. The relationships between the processes can then be modelled as follows:

$$(\overline{Y_1} \vee \overline{Y_2}) \cdot ((Y_1 \vee Y_2) \leftrightarrow Y_3) \tag{4.165}$$

The 0–1 variables can be used to model the satisfaction of the two clauses. Branching on these variables before the other variables is advantageous.

4.10.5 Other Applications

Truss design involves designing a structure, at minimum cost, to support objects of a given weight and size. Some bars in the structure are supported by other bars. Hence there is a *logical dependency* of the former on the latter which can be formulated using logical statements. The objective function will be represented as a linear combination of the literals.

Optimal pit limits involves finding the 'best' volume of earth and rock to extract from an open-cast mine in order to maximise profit. If the mine is 'discretised' by conceptually visualising the volume in discrete blocks there is a logical relationship between the blocks which should be excavated. In order to be able to extract the richer ore 'uneconomic' blocks above (overburden) must be extracted first. In total there will be an optimum combination of blocks which should be extracted to maximise total profit. An extension of the problem is to find the optimal order within

which to extract blocks in order to maximise discounted cash flow. Again one must observe the logical dependency between the blocks to be extracted.

Designing *safety precaution* measures, e.g. fire prevention, nuclear safety or sensor placement, can be formulated as a logical problem. Certain measures (e.g. locking or creating barriers and routine inspections) may or may not be implemented in conjunction with others. The detection of a potential hazardous event will depend on a logical combination of events. It may be desirable to express the logical relationship in as economical a manner as possible.

Database access languages (such as the widely used System Query Language (SQL)) allow the user to make a logical statement involving 'OR', 'AND' and 'NOT' concerning the combination of tables to look up when retrieving information. In order to make the search of very large databases as fast as possible (and not make mistakes in the specification) it is desirable to represent the logical statement as compactly as possible using a normal form. Therefore logical manipulation by the user, or the system being used, is desirable. *Data mining* is an area where these considerations are very relevant.

4.11 References and Further Work

As discussed in Chapter 2 the *satisfiability problem* is the standard problem used for comparison in defining a measure of *computational complexity* since many problems can be converted to it. The classic reference here is Garey and Johnson [41]. Pardalos et al. [88] edit a book devoted to the satisfiability problem.

Hooker [54] treats IP in the framework of *logical inference*. The *resolution* principle is due to Robinson [95] (in the form of statements in CNF). However, Quine [93, 94] was the first to define the method. He applied it to statements in DNF. Davis and Putnam [31] created the method under their names which was later extended by Loveland [75]. Jeroslow and Wang [65] give one of the best heuristics for choice of branching literal.

The representation of logical problems as IPs was presented early on by Williams [112].

Hooker [54, 53] discusses the similarities (but non-equivalence) of *prime implications* and *facets*.

Quine [95] discusses the problem of finding the *minimum sum of prime implicants* (the DNF form of finding the *minimum sum of prime implications*) but does not provide a viable method for other than very small problems.

Truemper [105] discusses the computational side of solving some of the problems in this chapter.

Horn clause systems form the basis of a number of logical languages used by *expert systems*. One of the earliest is PROLOG (see Kowalski [70]). Dowling and Gallier [34] describe how to solve the satisfiability problem for Horn clauses in linear time. Gallo and Scutella [40] discuss 'easy' satisfiability problems.

Constraint logic programming is described in many books such as Tsang [106], Van Hentenryck [108] and Milano [83]. Hooker [57] discusses the relationship with IP and lists many of the global predicates used. Yan and Hooker [128] discuss the formulation of the cardinality constraint as an IP and Williams and Yan [120] discuss the *all_different* constraint. Hybrid systems are discussed by Hooker [56].

Benders decomposition is due to Benders [14] and discussed in many textbooks such as Nemhauser and Wolsey [86] and Martin [80]. Geoffrion [42] describes a practical implementation of Benders decomposition.

Granot and Hammer [47] describe the conversion of PIP models to satisfiability problems and Hooker [55] gives the method of appending clauses as needed. An example of a 'difficult' pure 0–1 model (based on the British Petroleum/Shell demerger over a number of products and sites) and the value of appending 'cover' constraints are described in Williams [113].

Problems of *circuit design* are discussed in, for example, Weste and Harris [109]. Threshold functions are described by Muroga [84].

The *logical analysis of data* and its applications are described by many authors such as Boros et al. [19] and Anthony and Hammer [2] to name only a few.

Grossmann et al. [49] describe logical problems arising in *chemical engineering* and their solution aided by logical methods. Bollapragada et al. [17] describe the *truss design* problem. Williams [110], Jayawardane et al. [59] and Boland et al. [16] (among others) discuss the optimal pit limits problem.

The *Systems Query Language* (SQL) is described in Emerson et al. [36].

4.12 Exercises

4.12.1 Test if Example 4.1 is satisfiable using the DPL procedure.

4.12.2 Show by the application of resolution and absorption that the following statement is not satisfiable:

$$(X_1 \vee X_2) \cdot (\overline{X_1} \vee \overline{X_3} \vee \overline{X_4}) \cdot (\overline{X_2} \vee X_3) \cdot (\overline{X_2} \vee \overline{X_3}) \cdot (\overline{X_1} \vee \overline{X_3} \vee X_4)$$

4.12.3 Show that the application of the DPL procedure to a satisfiable statement in CNF must result in a satisfiable set of truth values.

4.12.4 Apply the DPL procedure to find a satisfiable set of truth values for

$$(X_1 \vee X_2 \vee \overline{X_3}) \cdot (X_1 \vee X_3 \vee X_4) \cdot (\overline{X_2} \vee X_3) \cdot (\overline{X_3} \vee \overline{X_4})$$

4.12.5 Convert the problem of determining the satisfiability of

$$(X_1 \vee \overline{X_2} \vee X_3) \cdot (X_1 \vee X_2 \vee X_4) \cdot (X_1 \vee \overline{X_2} \vee X_3) \cdot (X_1 \vee X_2 \vee \overline{X_4})$$

to an IP. Solve it by the branch-and-bound method (using the minimisation of the sum of variables as an arbitrary objective) starting with the LP relaxation. Use a suitable computer package if desired.

4.12.6 Before solving the model in 4.12.5 by branch-and-bound apply the constraint form of resolution and absorption to simplify the model. Contrast the LP relaxation with that of 4.12.5.

4.12.7 Show that, in general, resolution applied to disjunctive clauses is equivalent to the application of addition and rounding to the equivalent constraints (including, possibly, the 0–1 bounding constraints).

4.12.8 Show that adding logical constraints, with more than one common variable differing in sign, and possibly rounding (if necessary adding 0–1 constraints) results in a valid but redundant extra constraint.

4.12.9 Construct small 0–1 IP examples to show that facets do not exactly correspond to prime implications and vice versa.

4.12.10 Show that, in general, resolution applied to the pigeon hole problem will result in an exponential number of steps (as a function of the number of variables).

4.12.11 Show that input resolution only requires a linear number of steps (as a function of the number of variables).

4.12.12 Express constraints (4.59)–(4.67) as a conjunction of disjunctive clauses.

4.12.13 Create the minimum logical statement, in DNF, which represents the following table of truth values:

X_1	X_2	X_3	X_4
T	**F**	**T**	-
-	**T**	**F**	**F**
F	-	**T**	-

4.12.14 Create the minimum logical statement, in CNF, which represents the negation of that in 4.12.13.

4.12.15 Represent all the prime implications of the statement in 4.12.14 as the nodes of a graph. Draw directed arcs from each pair of prime implications whose consensus is another prime implication.

4.12.16 Model the condition $x \neq \{v_1, v_2, \ldots, v_n\}$ using IP constraints.

4.12.17 Model the predicate $all\_diff(x_1, x_2, \ldots, x_n\}$ using IP constraints in two different ways.

4.12.18 Represent the following statement:

$$(A \longrightarrow B) \longrightarrow [(B \longrightarrow (C \longrightarrow A)) \longrightarrow (B \longrightarrow A)]$$

as an electrical circuit where each literal corresponds to a switch.

4.12.19 Represent the statement in 4.12.18 by means of a net of NOR gates.

4.12.20 Formulate the problem of minimising the number of NOR gates to represent the 'exclusive OR' function as an IP. Solve this model (using a suitable package program if desired).

4.12.21 Design an electrical circuit that will only give a positive response when at least two out of five possible votes (registered by switches) of a committee are cast.

4.12.22 Formulate the general problem of minimising the number of threshold gates, each with three inputs, connected together to represent a logical statement in DNF as an IP.

4.12.23 Check that statements (4.156) and (4.160) are consistent (satisfiable).

4.12.24 Carry out all the steps of Example 4.6 using resolution and the DPL procedure.

4.12.25 Program the method described in Example 4.6 for solving PIP models as satisfiability problems using all-integer arithmetic in a suitable programming language.

4.12.26 Solve Example 4.6 using LP relaxations and 'logical cuts'. Use a suitable LP package.

References

1. Aho AV, Hopcroft JE, Ullman JD. The design and analysis of algorithms. Reading MA; Addison-Wesley; 1974.
2. Anthony M, Hammer PL. A Boolean measure of similarity. Discrete Appl Math, 2006; 154: 2242–2246.
3. Appa G, Pitsoulis L, Williams HP. Handbook of modelling for discrete optimization. New York: Springer; 2006.
4. Applegate DL, Bixby R, Chvátal V. The travelling salesman problem: a computational study. Princeton, NJ: Princeton University Press; 2006.
5. Balas E. Intersection cuts from disjunctive constraints. Management Research Report 330. Graduate School of Industrial Administration, Carnegie Mellon University; 1974.
6. Balas E. Facets of the knapsack polytope. Math Program 1975; 8: 146–164.
7. Balas E. Disjunctive programming. Ann Discrete Math 1979; 5: 3–51.
8. Balas E. A note on duality in discrete programming. J Optim Theory Appl 1979; 21: 573–578.
9. Balas E. Disjunctive programming: properties of the convex hull of feasible points. Discrete Appl Math, 1998; 89: 1–46.
10. Balas E, Ng SM. On the set covering polytope I: all facets in $\{0, 1, 2\}$. Math Program 1989; 43: 57–69.
11. Balas E, Padberg MW. On the set covering problem II. Oper Res 1975; 23: 74–80.
12. Barth P. Logic-based 0–1 constraint solving in constraint logic programming. Dordrecht: Kluwer; 1995.
13. Beale EML, Tomlin JA. Special facilities in a general mathematical programming symposium for non-convex functions. In: Lawrence J, editor. Proceedings of the 5th international conference in operational research. London: Tavistock Publications; 1970.
14. Benders JF. Partitioning procedures for solving mixed variables programming problems. Numer Math 1962; 4: 238–252.
15. Bland RG. New finite pivoting rules for the simplex method. Math Oper Res 1977; 2: 103–107.
16. Boland N, Dumitrescu I, Froyland G, Greixer A. LP-based dissaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. Computers and Operations Research, 2009; 36: 1064–1089.
17. Bollapragada S, Ghattas O, Hooker JN. Optimal design of truss structures by mixed logical and integer programming. Oper Res 2001; 49: 49–51.
18. Boole G. An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities. New York: Dover Publications; 1951. Original work published 1854.
19. Boros E, Hammer PL, Ibaraki T, Kogan A. Logical analysis of numerical data. Math Program 2000; 79: 163–190.

20. Bradley GH. Transformation of integer programs to knapsack problems. Discrete Math 1971; 1: 29–45.
21. Carré BA. An algebra for network routing problems. J Inst Math Appl 1971; 7: 273–293.
22. Chandru V, Hooker JN. Optimization methods for logical inference. New York: Wiley; 1999.
23. Chvátal V. Edmonds polytopes and a heirarchy of combinatorial problems. Discrete Math 1973; 4: 305–337.
24. Chvátal V. Linear programming. Freeman: 1983.
25. Claus A. A new formulation for the travelling salesman problem. SIAM J Algebra Discrete Math 1984; 5: 21–25.
26. Cook SA. The complexity of theorem-proving procedures 1. Proceedings of the third annual ACM symposium on the theory of computing; 1971. P. 151–158.
27. Cornuejols G, Sassano A. On the 0–1 facets of the set covering polytope. Math Program B 1989; 45: 45–56.
28. Cunningham Green RA. Minimax algebra, vol. 166. Lecture notes in economics and mathematical systems. Springer Verlag; Berlin 1979.
29. Dakin RJ. A tree search algorithm for mixed integer programming. Comput J 1965; 8: 250–255.
30. Dantzig GB. Linear programming and extensions. Princeton: Princeton University Press; 1963.
31. Davis M, Putnam H. A computing procedure for quantification theory. J Assoc Comput Mach 1960; 7: 201–215.
32. Day RE, Williams HP. MAGIC: the design and use of an interactive modelling language for integer programming. IMA J Math Manage 1986/87; 1: 53–65.
33. Dietricht BL, Escudero LF, Chance F. Efficient reformulation of 0-1 programs: methods and computational results. Discrete Appl Math 1993; 42: 147–175.
34. Dowling WF, Gallier JH. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. Journal of Logic Programming 1984; 1: 267–284.
35. Edmonds J, Johnson E. Matching: a well-solved class of integer linear programmes. In Guy RK, editor. Combinatorial structures and their applications. New York: Gordon and Breach; 1970. P. 88–92.
36. Emerson SL, Darnovski M, Bowman JS. The practical SQL handbook. 3rd ed. Emeryville, CA: Sybase; 1996.
37. Fourer R, Gay DM, Kernigan BW. AMPL: a modelling language for mathematical programming. Pacific Grove, CA: Thomson; 2003.
38. Fourier JBJ. Solution d'une question du calcul des inegalites. Oeuvres II. Paris; 1826. P. 317–328.
39. Frege G. In: Geach P, Black M, editors. Translations from the Philosophical Writings of Frege. Oxford; 1952.
40. Gallo G, Scutella MG. Polynomially solvable satisfiability problems. Inf Process Lett 1988; 29: 221–227.
41. Garey MR, Johnson DS. Computers and intractibility San Francisco: W.H. Freeman; 1979.
42. Geoffrion AM. Lagrangian relaxation for integer programming. Math Program Study 1974; 2: 82–114.
43. Geoffrion AM, Marsten RE. Integer programming algorithms: a framework and state of the art survey. Manage Sci 1972; 18: 465–491.
44. Gödel G. On undecidable propositions of formal mathematical systems. Princeton, NJ: Princeton University Press; 1934.
45. Gomory RE. An all-integer programming algorithm. In Muth JF, Thomson GL, editors. Industrial scheduling. Englewood Cliffs, NJ: Prentice-Hall; 1963. P. 193–206.
46. Gondzio J, Grothey A. Direct solution of linear systems of size 10^9 arising in optimization and interior point methods. In: Wyrzykowski R, Dongarra J, Meyer N, Wasnieski J, editors. Parallel processing and applied mathematics PPAM 2005. lecture notes in computer science, vol. 3911. New York: Springer-Verlag; 2006. P. 513–525.

47. Granot F, Hammer PL. On the use of boolean functions in 0–1 programming. Meth Oper Res, 1971; 12: 154–184.
48. Greenberg HJ, Murphy FH. A comparison of mathematical programming modelling systems. Ann Oper Res 1992; 38: 177–238.
49. Grossmann IE, Hooker JN, Raman R, Yan H. Logic cuts for processing networks with fixed charges. Comput Oper Res 1994; 21: 265–279.
50. Hadjiconstantinou E, Lucas C, Mitra G, Moody S. Tools for reformulating logical forms into zero-one mixed integer programs.
Eur J Oper Res 1994; 72: 263–277.
51. Hall HAJ, McKinnon KIM. The simplest example where the simplex method cycles and where EXPAND fails to prevent cycles. Math Program 2004; 100(1): 133–150.
52. Hammer PL, Johnson EL, Peled UN. Facets of regular 0–1 polytopes. Math Program 1975; 8: 179–206.
53. Hooker JN. Generalized resolution and cutting planes. Ann Oper Res 1986; 12: 217–239.
54. Hooker JN. A quantitative approach to logical inference. Decis Support Syst 1988; 4: 45–69.
55. Hooker JN. Solving the incremental satisfiability problem. J Logic Program 1993; 15: 177–186.
56. Hooker JN. Logic-based methods of optimization. New York: Wiley; 2000.
57. Hooker JN. Integrated methods for optimization. New York: Springer; 2007.
58. Hürlimann T. IP, MIP and logical modelling using LPL. Paper presented at Symposium on applied mathematical modelling and optimisation, Budapest, January 1993.
59. Jayawardane AK, King M, Wilson JM. Mathematically modelling the problem of earth moving. Bull Inst Math Appl 1993; 29: 16–23.
60. Jeroslow RG. There cannot be any algorithm for integer programming with quadratic constraints. Oper Res 1972; 21: 221–224.
61. Jeroslow RG. The simplex algorithm with the pivot rule of maximising criterion improvement. Discrete Math 1973; 4: 367–377.
62. Jeroslow RG. Representability in mixed integer programming 1: characterisation results. Discrete Appl Math 1987; 17: 223–243.
63. Jeroslow RG. Logic based decision support: mixed integer model formulation. Annals of discrete mathematics, vol. 40. Amsterdam: North-Holland; 1989 (complete volume devoted to this monograph).
64. Jeroslow RG, Lowe JK. Experimental results with the new techniques for integer programming formulation. J Oper Res Soc 1985; 36: 393–403.
65. Jeroslow RG, Wang J. Solving propositional satisfiability problems. Ann Math AI 1990; 1: 167–187.
66. Karmarkar N. A new polynomial algorithm for linear programming. Combinatorica 1984; 4: 375–395.
67. Karp RM. Reducability among combinatorial problems. In: Miller RW, Thatcher JW, editors. Complexity in computer computations. New York: Plenum Press; 1972. P. 85–103.
68. Khachian LG. A polynomial algorithm for linear programming. Dokl Akad Nauk SSSR 1979; 244: 1093–1096.
69. Klee V, Minty GJ. How good is the simplex algorithm. In: Shisha O, editor. Inequalities III. London: Academic Press; 1972. P. 159–175.
70. Kowalski R. Prolog as a logic programming language. In: Proceedings of AICA congress, Paria, Italy, 1981. P. 1029–1034.
71. Land AH, Doig A. An automatic method of solving discrete programming problems. Econometrica 1960; 28: 497–520.
72. Langer SK. An introduction to symbolic logic. 2nd ed. New York: Dover Publications; 1953.
73. Langford CH. Some theorems of deducibility. Ann Math I 1927; 28: 16–40.
74. Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB, editors. The travelling salesman problem. Chichester: Wiley; 1995.

75. Loveland DW. Automated theorem proving: a logical basis. Amsterdam: North-Holland; 1978.
76. Marchand H, Wolsey LA. Aggregation and mixed integer rounding to solve MIPs. Oper Res 2001; 49: 363–371.
77. Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.
78. Martin RK. Generating alternative mixed-integer programming models using variable redefinition. Oper Res 1984; 35: 820–831.
79. Martin RK. Large scale linear and integer programming: a unified approach. Boston: Kluwer; 1999.
80. McKinnon KIM, Williams HP. Constructing integer programming models by the predicate calculus. Ann Oper Res 1989; 21: 227–246.
81. Mendelson E. Introduction to mathematical logic. Princeton, NJ: Van Nostrand; 1964.
82. Meyer RR. Integer and mixed integer programming models. J Opt Theory Appl 1975; 16: 191–206.
83. Milano M, editor. Constraint and integer programming: toward a unified methodolgy. Dordrecht: Kluwer 2004.
84. Muroga S. Threshold logic and its applications. New York: Wiley; 1971.
85. Nagel E, Newman JR. Godel's proof. London, UK: Routledge and Kegan Paul; 1959.
86. Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. New York: Wiley; 1988.
87. Orman AJ, Williams HP. A survey of different integer programming formulations of the travelling salesman problem. In: Gatu C, Kontoghiorghes E, editors. Advances in computational management science 9 optimisation, econometric and financial analysis (2006). Berlin: Springer; 2006.
88. Pardalos PM, Du DZ, Gu J, editors. The satisfiability problem. DIMACS Series, 13. American Mathematical Society; Providence, Rhode Island, 1998.
89. Peirce CS. A Boolean algebra with one constant. In: Hartshome L, Weiss P, editors. Collected papers of Charles Sanders Peirce, vol. 4. Harvard University Press; 1931–35. P. 12–20.
90. Post E. Introduction to a general theory of elementary propositions. Am J Math 1921; 1: 163–185.
91. Presburger M. Uebe ndikeit eines gewissen systems der arithmetik ganzer zahlen in welchem die addition als einzige operation hervortritt. Comptes Rendus, I Congres des Pays Slaves, Warsaw, 1929. P. 192–201.
92. Pulleyblank W. Faces of matching polyhedra, PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1973.
93. Quine WV. The problem of simplifying truth functions. Am Math Mon 1952; 59: 521–531.
94. Quine WV. A way to simplify truth functions. Am Math Mon 1955; 62: 627–631.
95. Robinson JA. A machine orientated logic based on the resolution principle. J Assoc Comput Mach 1965; 12: 23–41.
96. Russell B, Whitehead AN. Principia mathematica, vols. I–III. Cambridge: Cambridge University Press; 1910–13.
97. Ryan DM. The solution of massive generalised set partitioning problems in aircrew rostering. J Oper Res Soc 1992; 43: 459–462.
98. Schrijver A. Theory of linear and integer programming. Chichester, UK: Wiley; 1856.
99. Sheffer HM. A set of five independent postulates for Boolean algebras with applications to logical constants. Trans Am Math Soc 1913; 14: 481–488.
100. Sherali HD, Shetty CM. Optimization with disjunctive constraints. Berlin; Springer-Verlag: 1980.
101. Shoenfield JR. Mathematical logic. Reading, MA: Addison-Wesley; 1967.
102. Smale S. On the average speed of the simplex method of linear programming. Math Program 1983; 27: 241–262.

103. Spencer-Brown G. Laws of Form. New York: Dutton; 1979.
104. Tind J, Wolsey LA. An elementary survey of general duality theory in mathematical programming. Math Program 1981; 21: 241–261.
105. Truemper K. Effective logic computation. New York: Wiley; 1998.
106. Tsang E. Foundations of constraint satisfaction. London: Academic Press; 1983.
107. Tseitin GS. On the complexity of derivation in the propositional calculus. In: Slisenko AO, editor. Structures in constructive mathematics and mathematical logic, part II. Steklov Mathematical Institute; 1968.
108. Van Hentenryck P. Constraint satisfaction in logic programming. Cambridge, MA: MIT Press; 1989.
109. Weste NHE, Harris D. CMOS VLSI design: a circuits and system perspective. 3rd ed. Reading, MA: Pearson, Addison-Wesley; 2004.
110. Williams HP. Experiments in the formulation of integer programming problems. Math Program Stud 1974; 2: 180–197.
111. Williams HP. Fourier–Motzkin elimination extended to integer programming problems. J Comb Theory 1976; 21:118–123.
112. Williams HP. Logical problems and integer programming. Bull Inst Math Appl 1977; 13: 18–25.
113. Williams HP. The reformulation of two mixed integer programming problems. Math Program 1978; 14: 325–331.
114. Williams HP. Fourier's method of linear programming and its dual. Am Math Mon 1986; 93: 681–695.
115. Williams HP. Linear and integer programming applied to the propositional calculus. Int J Syst Res Inf Sci 1987; 2: 81–100.
116. Williams HP. An alternative explanation of disjunctive formulations. Eur J Oper Res 1994; 72: 200–203.
117. Williams HP. Logic applied to integer programming and integer programming applied to logic. Eur J Oper Res 1995; 81: 605–616.
118. Williams HP. Duality in mathematics and linear and integer programming. J Optim Theory Appl 1996; 90: 257–278.
119. Williams HP. Model building in mathematical programming. 4th ed. Chichester: Wiley; 1999.
120. Williams HP, Yan H. Representations of the all-different predicate of constraint satisfaction in integer programming. INFORMS J Comput 2001; 13: 96–103.
121. Wilson JM. A method of (0–1) programming using a Boolean approach. In: Bothey TB, editor. Proceedings of CP77. Liverpool: University of Liverpool; 1977.
122. Wilson JM. Boolean simplification and integer inequalities. Comput J 1977; 20: 356–358.
123. Wilson JM. Compact normal forms in propositional logic and integer programming formulations. Comput Oper Res 1990; 90: 309–314.
124. Wittgenstein L. Tractatus logico_philosophicus, translated by Pears DF, McGuinness BF. New York: Humanities Press; 1961.
125. Wolsey LA. Faces for a linear inequality in 0–1 variables. Math Program 1975; 8: 165–178.
126. Wolsey LA. Strong formulations for mixed integer programming: a survey. Math Program 1989; 45: 173–191.
127. Wong RT. Integer programming formulations of the travelling salesman problem. Proceedings of IEEE conference on circuits and computers; Port Chester, New York, 1980. 149–152.
128. Yan H, Hooker JN. Tight representations of logical constraints as cardinality rules. Math Program 1995; 85: 363–377.

Index

0-1 integer programming models, 47
0-1 variables, 35, 49, 71, 73, 88

A
absorption, 106, 107, 109, 111, 118
Aho, A.V., 67, 145
algebraic modelling languages, 96
algorithm, 3
all-different constraint, 125
all-different predicate, 142
alternative optima of a linear programme, 31
AMPL modelling language, 97
analytical truth, 1
Anthony, M., 142, 145
Appa, G., 67, 145
Applegate, D.L., 67, 145
arithmetic without multiplication, 1, 20, 22, 36
assignment problem, 35, 125
at-least constraint, see also greater-than-or-equal constraint, 125
at-least-predicate, 98
atomic statements, 3
axioms, 1

B
Balas, E., 67, 102, 145
bandwidth allocation, 58
Barth, P., 102, 145
Beale, E.M.L., 67, 145
Benders decomposition, 128, 142
Benders, J.F., 142, 145
binary variables, see 0-1 variables, 51
Bixby, R., 67, 145
Black, M., 146
Bland, R.G., 67, 145
blending, 34, 52
Boland, N., 142, 145
Bollapragada, S., 142, 145
Boole, G., 22, 145
boolean algbra, see propositional calculus, 3
Boros, E., 142, 145
Bowman, J.S., 142, 146
Bradley, G.H., 68, 146
branch and bound, 68, 71, 112, 140
branch and bound algorithm, 38, 41
branch and cut, 45

C
calculus of indications, 11, 22
cardinality constraint, 125
Carré, B.A., 102, 146
Chance, F., 102, 146
Chandru, V., 102, 146
chemical processing, 34, 139, 142
Chvátal cuts, 47
Chvátal function, 47
Chvátal rank, 54, 66, 69
Chvátal rank of an integer programme, 47
Chvátal, V., 66, 67, 145, 146
circuit constraint, 126
Claus, A., 146
column generation, 128
complete connectives, 10, 23
complete sets of connectives, 10
complete sum of prime implications, 108
completeness of a system, 3
compound statements, 3
computational complexity, 10, 25, 35, 58, 60, 141
computer circuit design, 54
conclusion of an argument, 2
conjunctive normal form, 6, 9, 85, 87, 89, 106, 118, 119
connective arrow, 10, 22, 135
connectives, 4, 12, 71, 73, 97
consensus of two clauses, 118
consistency of a system, 1, 2, 105
constraint logic programming, 97, 99, 123, 142

constraint propogation, 127
constraint satisfaction, see constraint logic programming, 123
contradiction, 4, 6
contrapositive, 6, 72
convex feasible regions, 75
convex hull of a disjunction of polytopes or polyhedra , 91
convex hull of an integer programme, 43, 47, 85
convex models, 57
convexification of an integer programming model, 91
Cook, S.A., 67, 146
Cornuejols, G., 67, 146
credit scoring, 137
crew scheduling, 54, 128
Cunninghame Green, R.A., 102, 146
cutting planes, 44, 102, 112
cutting stock problem, 58, 128

D
Dakin, R.J., 67, 146
Dantzig, G.B., 66, 146
Darnovski, M., 142, 146
data base access languages, 141
data compression, 137
data mining, 141
data structures, 61
Davis, M., 141, 146
Davis-Putnam-Loveland procedure, 109, 129
Day, R.E., 102, 146
De Morgan's laws, 5, 8, 9, 15, 20, 99, 105, 119
decidable fragments of mathematics, 18
decision procedure, 3, 18, 20, 36
Dietricht, B.L., 102, 146
discrete optimisation, 2
disease diagnosis, 138
disjunctive formulations, 92
disjunctive normal form, 6, 9, 22, 85, 86, 89, 95, 118, 119, 138
disjunctive programming, 75, 79, 102
dissaggregating constraints, 92
distribution, 34
Doig, A., 67, 147
domain of a variable, 15, 124, 127
Dowling, W.F., 141, 146
Du, D.Z., 141, 148
dual of a disjunctive programme, 96, 103
dual of a linear programme, 28, 90, 95
dual of an integer programme, 89
dual, logical, 5, 9
duality gap, 30, 46, 86
duality theorem of linear programming, 30
Dumitrescu, I., 142, 145
dynamic presolve, 128

E
Edmonds, J., 67, 146
electrical circuit design, 10, 134, 142, 143
elimination of existential quantifiers, 19, 26, 34
Emerson, S.L., 142, 146
epigraph, 84
equivalent logical statements, 116
Escudero, L.F., 102, 146
exclusive or, 136
existential quantifier, 19, 21
expert systems, 141
exponential algorithms, 62
exponential complexity, 61
extended conjunctive normal form, 8
extended disjunctive normal form, 7, 9, 14
extreme rays, 82
extreme rays of a polyhedron, 34, 93, 94

F
facets of a polytope or polyhedron, 30, 44, 46, 143
facility location problem, 50, 51, 83
feasibility problems, 65
feasible region of a linear programme, 30, 43
feasible solution of a linear programme, 27
first order theory, see predicate calculus, 16
fixed charge problem, 51, 83
fixed costs, 50, 51
flip-flop, 136
Fourer, R., 102, 146
Fourier, J.B.J., 66, 146
Fourier-Motzkin elimination, 28, 66, 123
fragments of mathematics, 1
Frege, E., 22, 146
Froyland, G., 142, 145
full dimensional polytopes and polyhedra, 30
functions, 18

G
Gallier, J.H., 141, 146
Gallo, G., 141, 146
Garey, M.R., 67, 141, 146
Gay, D.M., 102, 146
Geach, P., 146
generalised set covering problem, 110
Geoffrion, A.M., 67, 142, 146
Ghattas, O., 142, 145
Gleixner, A., 142
global constraints, 124
global optimisation, 55

Gödel, G., 1, 22, 64, 68, 146
Gomory, R.E., 67, 146
Gondzio, J., 67, 146
Granot, F., 142, 147
greater-than-or-equal predicate, 98
Greenberg, H.J., 102, 147
Greixer, A., 145
Grossmann, I.E., 142, 147
Grothey, A., 67, 146
Gu, J., 141, 148

H
Hürlimann, T., 102
Hadjiconstantinou, E., 102, 147
Hall, H.A.J., 67, 147
Hammer, P.L., 67, 142, 145, 147
Harris, D., 142
heuristics, 38, 65
Hooker, J.N., 102, 141, 142, 145–147, 149
Hopcroft, J.E., 67, 145
horn clauses, 119, 141
Hürlimann, T., 147
hybrid systems, 128, 142

I
Ibaraki, T., 142, 145
index sets, 96
infeasible linear programmes, 34, 69
inference, 2, 141
inference problems, 2, 46, 105
input resolution, 120, 121, 143
instantiation, 15
integer programming, 2, 6, 25, 35
interpretation, 1
intractible problems, 66
investment problems, 60

J
Jayawardane, A.K., 142, 147
Jeroslow, R.G., 67, 68, 102, 141, 147
Johnson, D.S., 67, 141, 146
Johnson, E.L., 67, 146, 147

K
Kachian, L.G., 67, 147
Karmarkar, N., 67, 147
Karp, R.M., 67, 147
Kernigan, B.W., 102, 146
King, M., 142, 147
Klee, V., 67, 147
knapsack problem, 58, 63, 70, 137
Kogan, A., 142, 145
Kowalski, R., 141, 147

L
Land, A.H., 67, 147
Langer, S.K., 22, 147
Langford, C.H., 22, 147
Lawler, E.L., 67, 147
Lenstra, J.K., 67, 147
linear programming, 25
linear programming relaxation, 143
linear programming resolution, 115
local constraints, 124
locally optimal solutions, 55
logical analysis of data (LAD), 137, 142
logical circuit design, 119
logical dual, 30, 119
logical nets, design of, 135
logical resolution, 47
logical simplification, 108
Loveland, D.W., 141, 148
Lowe, J.K., 102, 147
Lucas, C., 102, 147

M
Marchand, H., 67, 148
Marsten, R.E., 67, 146
Martello, S., 67, 148
Martin, R.K., 66, 102, 142, 148
matching problem, 54, 60
maximum satisfiability problem, 107, 113, 114
McKinnon, K.I.M., 67, 102, 147, 148
Mendelson, E., 22, 148
meta relation, 4
meta symbols, 12
meta system, 1, 2, 64
Meyer, R.R., 102, 148
Milano, M., 142, 148
minimax algebra, 90
minimum cost network flow, 35, 102
minimum logical statement, 143
minimum size logical statement, 138, 143
Minty, G.J., 67, 147
MIP representability, 71, 78, 96, 100
Mitra, G., 102, 147
mixed integer programming, 45
Moody, S., 102, 147
Muroga, S., 142, 148
Murphy, F.H., 102, 147

N
Nagel, E., 22, 68, 148
nand, 10, 13
Nemhauser, G.L., 67, 142, 148
nested predicates, 98
Newman, J.R., 22, 68, 148
Ng, S.M., 67, 145

non deterministic computer, 64
non-convex feasible regions, 75
non-convex models, 57
non-input resolution, 121
non-linear programming, 55, 75
nor, 10, 135
NOR gates, 135, 144
normal forms, 5, 98, 136
not-equal predicate, 124
NP complete, 64
NP hard, 65

O
objective function of a linear programme, 26, 31
oil refinery scheduling, 34
optimal pit limits problem, 140
optimisation problems, 2, 3, 65
Orman, A.J., 67, 148

P
Padberg, M.W., 67, 145
Pardalos, P.M., 141, 148
pattern recognition, 137
Peirce, C.S., 22, 148
Peled, U.N., 67, 147
piecewise linear functions, 55
pigeon hole problem, 113, 143
Pitsoulis, L., 67, 145
polyhedron, 30
polynomial algorithms, 62
polynomial complexity, 61
polytope, 30
Post, E., 22, 148
predicate calculus, 3, 14, 15, 22, 23, 25, 26, 68, 105
predicates, 14
premises of an argument, 2
prenex normal form, 16
Presburger, M., 22, 148
primal of a linear programme, 29
prime implicants, 118, 119, 141
prime implications, 108, 112, 116, 141, 143
production, 34
products of variables, 53
projection of polytopes and polyhedra, 34
propositional calculus, 3, 14, 15, 22, 84, 102, 105
provability of a statement, 1
Pulleyblank, W., 67, 148
Putnam, H., 141, 146

Q
quadratic assignment problem, 60
quantifiers, 15, 97
Quine, W.V., 141, 148

R
Raman, R., 142, 147
recession directions of a polyhedron, 82, 84
reduction of a model, see also presolve, 112, 124, 127
relations, 18
relaxation of a linear programme, 38, 69
relaxation of an integer programme, 48, 59, 89
resolution, 106, 109, 111, 120, 141, 143
resolvent of two logical clauses, 107
Rinnooy Kan, A.H.G., 67, 147
Robinson, J.A., 141, 148
routing problems, 58
rules of deduction, 1
Russell, B., 1, 22, 148
Ryan, D.M., 67, 148

S
safety systems, design of, 137, 141
Sassano, A., 67, 146
satisfiability problem, 60, 65, 105, 109, 129, 141
scheduling problems, 60
Schmoys, D.B., 147
Schrijver, A., 67, 148
scope of a quantifier, 16
Scutella, M.G., 141, 146
separable functions, 57
set covering problem, 53, 110, 128
set packing problem, 53
set partitioning problem, 53
set theory, 1, 13
sharp formulations, 85, 95
sharp integer programming formulations, 44
Sheffer stroke, 10, 13, 22
Sheffer, H.M., 22, 148
Sherali, H.D., 102, 148
Shetty, C.M., 102, 148
Shmoys, D.B., 67
Shoenfield, J.R., 22, 148
simplest equivalent logical statement, 108
simplex algorithm, 61
simplification of logical statements, 9, 54, 116
Smale, S., 67, 148
space complexity, 66
Spencer-Brown, G., 22, 149
splitting of variables, 92, 102
SQL language, 141
standard form of a linear programme, 26
subtours of a travelling salesman problem, 59
symbolic logic, 2

systems query language (SQL), 142

T
tautology, 4, 6, 22
telecommunications networks, 128
theory of dense linear order, 1, 18, 22
threshold gates, 137, 144
time complexity, 66
Tind, J., 67, 149
Tomlin, J.A., 67, 145
Toth, P., 67, 148
transportation problem, 35
travelling salesman problem, 44, 47, 58, 102, 126
Truemper, K., 141, 149
truss design, 140, 142
truth of a statement, 1
truth tables, 4, 6, 22, 106, 116
Tsang, E., 142, 149
Tseitin, G.S., 22, 149

U
Ullman, J.D., 67, 145
unbounded Chvátal rank, 47
unbounded linear programmes, 30, 34, 69, 82
unsolvable problems, 64, 66

V
Van Hentenryck, P., 142, 149
Venn diagrams, 13, 23, 75
vertices of a polytope or polyhedron, 30, 34

W
Wang, J., 141, 147
Weste, N.H.E., 142
Whitehead, A.N., 1, 22, 148
Williams, H.P., 66, 67, 102, 141, 142, 145, 146, 148, 149
Wilson, J.M., 22, 102, 142, 147, 149
Wittgenstein, L., 22, 149
Wolsey, L.A., 67, 102, 142, 148, 149
Wong, R.T., 102, 149

X
Yan, H., 142, 147, 149

Early titles in the
INTERNATIONAL SERIES IN
OPERATIONS RESEARCH & MANAGEMENT SCIENCE
Frederick S. Hillier, Series Editor, *Stanford University*

Saigal/ *A MODERN APPROACH TO LINEAR PROGRAMMING*
Nagurney/ *PROJECTED DYNAMICAL SYSTEMS & VARIATIONAL INEQUALITIES WITH APPLICATIONS*
Padberg & Rijal/ *LOCATION, SCHEDULING, DESIGN AND INTEGER PROGRAMMING*
Vanderbei/ *LINEAR PROGRAMMING*
Jaiswal/ *MILITARY OPERATIONS RESEARCH*
Gal & Greenberg/ *ADVANCES IN SENSITIVITY ANALYSIS & PARAMETRIC PROGRAMMING*
Prabhu/ *FOUNDATIONS OF QUEUEING THEORY*
Fang, Rajasekera & Tsao/ *ENTROPY OPTIMIZATION & MATHEMATICAL PROGRAMMING*
Yu/ *OR IN THE AIRLINE INDUSTRY*
Ho & Tang/ *PRODUCT VARIETY MANAGEMENT*
El-Taha & Stidham/ *SAMPLE-PATH ANALYSIS OF QUEUEING SYSTEMS*
Miettinen/ *NONLINEAR MULTIOBJECTIVE OPTIMIZATION*
Chao & Huntington/ *DESIGNING COMPETITIVE ELECTRICITY MARKETS*
Weglarz/ *PROJECT SCHEDULING: RECENT TRENDS & RESULTS*
Sahin & Polatoglu/ *QUALITY, WARRANTY AND PREVENTIVE MAINTENANCE*
Tavares/ *ADVANCES MODELS FOR PROJECT MANAGEMENT*
Tayur, Ganeshan & Magazine/ *QUANTITATIVE MODELS FOR SUPPLY CHAIN MANAGEMENT*
Weyant, J./ *ENERGY AND ENVIRONMENTAL POLICY MODELING*
Shanthikumar, J.G. & Sumita, U./ *APPLIED PROBABILITY AND STOCHASTIC PROCESSES*
Liu, B. & Esogbue, A.O./ *DECISION CRITERIA AND OPTIMAL INVENTORY PROCESSES*
Gal, T., Stewart, T.J., Hanne, T. / *MULTICRITERIA DECISION MAKING: Advances in MCDM Models, Algorithms, Theory, and Applications*
Fox, B.L. / *STRATEGIES FOR QUASI-MONTE CARLO*
Hall, R.W. / *HANDBOOK OF TRANSPORTATION SCIENCE*
Grassman, W.K. / *COMPUTATIONAL PROBABILITY*
Pomerol, J-C. & Barba-Romero, S./ *MULTICRITERION DECISION IN MANAGEMENT*
Axsäter, S. / *INVENTORY CONTROL*
Wolkowicz, H., Saigal, R., & Vandenberghe, L. / *HANDBOOK OF SEMI-DEFINITE PROGRAMMING: Theory, Algorithms, and Applications*
Hobbs, B.F. & Meier, P. / *ENERGY DECISIONS AND THE ENVIRONMENT: A Guide to the Use of Multicriteria Methods*
Dar-El, E. / *HUMAN LEARNING: From Learning Curves to Learning Organizations*
Armstrong, J.S. / *PRINCIPLES OF FORECASTING: A Handbook for Researchers and Practitioners*
Balsamo, S., Personé, V., & Onvural, R./ *ANALYSIS OF QUEUEING NETWORKS WITH BLOCKING*
Bouyssou, D. et al. / *EVALUATION AND DECISION MODELS: A Critical Perspective*
Hanne, T. / *INTELLIGENT STRATEGIES FOR META MULTIPLE CRITERIA DECISION MAKING*
Saaty, T. & Vargas, L. / *MODELS, METHODS, CONCEPTS and APPLICATIONS OF THE ANALYTIC HIERARCHY PROCESS*
Chatterjee, K. & Samuelson, W. / *GAME THEORY AND BUSINESS APPLICATIONS*
Hobbs, B. et al. / *THE NEXT GENERATION OF ELECTRIC POWER UNIT COMMITMENT MODELS*
Vanderbei, R.J. / *LINEAR PROGRAMMING: Foundations and Extensions, 2nd Ed.*
Kimms, A. / *MATHEMATICAL PROGRAMMING AND FINANCIAL OBJECTIVES FOR SCHEDULING PROJECTS*
Baptiste, P., Le Pape, C. & Nuijten, W. / *CONSTRAINT-BASED SCHEDULING*

Early titles in the
INTERNATIONAL SERIES IN
OPERATIONS RESEARCH & MANAGEMENT SCIENCE
(Continued)

Feinberg, E. & Shwartz, A. / *HANDBOOK OF MARKOV DECISION PROCESSES*: *Methods and Applications*
Ramík, J. & Vlach, M. / *GENERALIZED CONCAVITY IN FUZZY OPTIMIZATION AND DECISION ANALYSIS*
Song, J. & Yao, D. / *SUPPLY CHAIN STRUCTURES: Coordination*, *Information and Optimization*
Kozan, E. & Ohuchi, A. / *OPERATIONS RESEARCH/ MANAGEMENT SCIENCE AT WORK*
Bouyssou et al. / *AIDING DECISIONS WITH MULTIPLE CRITERIA: Essays in Honor of Bernard Roy*
Cox, Louis Anthony, Jr. / *RISK ANALYSIS: Foundations*, *Models and Methods*
Dror, M., L'Ecuyer, P. & Szidarovszky, F. / *MODELING UNCERTAINTY: An Examination of Stochastic Theory, Methods, and Applications*
Dokuchaev, N. / *DYNAMIC PORTFOLIO STRATEGIES: Quantitative Methods and Empirical Rules for Incomplete Information*
Sarker, R., Mohammadian, M. & Yao, X. / *EVOLUTIONARY OPTIMIZATION*
Demeulemeester, R. & Herroelen, W. / *PROJECT SCHEDULING: A Research Handbook*
Gazis, D.C. / *TRAFFIC THEORY*
Zhu/ *QUANTITATIVE MODELS FOR PERFORMANCE EVALUATION AND BENCHMARKING*
Ehrgott & Gandibleux/ *MULTIPLE CRITERIA OPTIMIZATION: State of the Art Annotated Bibliographical Surveys*
Bienstock/ *Potential Function Methods for Approx. Solving Linear Programming Problems*
Matsatsinis & Siskos/ *INTELLIGENT SUPPORT SYSTEMS FOR MARKETING DECISIONS*
Alpern & Gal/ *THE THEORY OF SEARCH GAMES AND RENDEZVOUS*
Hall/ *HANDBOOK OF TRANSPORTATION SCIENCE - 2nd Ed.*
Glover & Kochenberger/ *HANDBOOK OF METAHEURISTICS*
Graves & Ringuest/ *MODELS AND METHODS FOR PROJECT SELECTION: Concepts from Management Science, Finance and Information Technology*
Hassin & Haviv/ *TO QUEUE OR NOT TO QUEUE: Equilibrium Behavior in Queueing Systems*
Gershwin et al/ *ANALYSIS & MODELING OF MANUFACTURING SYSTEMS*
Maros/ *COMPUTATIONAL TECHNIQUES OF THE SIMPLEX METHOD*
Harrison, Lee & Neale/ *THE PRACTICE OF SUPPLY CHAIN MANAGEMENT: Where Theory and Application Converge*
Shanthikumar, Yao & Zijm/ *STOCHASTIC MODELING AND OPTIMIZATION OF MANUFACTURING SYSTEMS AND SUPPLY CHAINS*
Nabrzyski, Schopf & Weglarz/ *GRID RESOURCE MANAGEMENT: State of the Art and Future Trends*
Thissen & Herder/ *CRITICAL INFRASTRUCTURES: State of the Art in Research and Application*
Carlsson, Fedrizzi, & Fullér/ *FUZZY LOGIC IN MANAGEMENT*
Soyer, Mazzuchi & Singpurwalla/ *MATHEMATICAL RELIABILITY: An Expository Perspective*
Chakravarty & Eliashberg/ *MANAGING BUSINESS INTERFACES: Marketing*, *Engineering*, *and Manufacturing Perspectives*
Talluri & van Ryzin/ *THE THEORY AND PRACTICE OF REVENUE MANAGEMENT*
Kavadias & Loch/*PROJECT SELECTION UNDER UNCERTAINTY: Dynamically Allocating Resources to Maximize Value*
Brandeau, Sainfort & Pierskalla/ *OPERATIONS RESEARCH AND HEALTH CARE: A Handbook of Methods and Applications*
Cooper, Seiford & Zhu/ *HANDBOOK OF DATA ENVELOPMENT ANALYSIS: Models and Methods*
Luenberger/ *LINEAR AND NONLINEAR PROGRAMMING*, *2nd Ed.*

Early titles in the
INTERNATIONAL SERIES IN
OPERATIONS RESEARCH & MANAGEMENT SCIENCE
(Continued)

Sherbrooke/ *OPTIMAL INVENTORY MODELING OF SYSTEMS: Multi-Echelon Techniques, Second Edition*
Chu, Leung, Hui & Cheung/ *4th PARTY CYBER LOGISTICS FOR AIR CARGO*
Simchi-Levi, Wu & Shen/ *HANDBOOK OF QUANTITATIVE SUPPLY CHAIN ANALYSIS: Modeling in the E-Business Era*
Gass & Assad/ *AN ANNOTATED TIMELINE OF OPERATIONS RESEARCH: An Informal History*
Greenberg/ *TUTORIALS ON EMERGING METHODOLOGIES AND APPLICATIONS IN OPERATIONS RESEARCH*
Weber/ *UNCERTAINTY IN THE ELECTRIC POWER INDUSTRY: Methods and Models for Decision Support*
Figueira, Greco & Ehrgott/ *MULTIPLE CRITERIA DECISION ANALYSIS: State of the Art Surveys*
Reveliotis/ *REAL-TIME MANAGEMENT OF RESOURCE ALLOCATIONS SYSTEMS: A Discrete Event Systems Approach*
Kall & Mayer/ *STOCHASTIC LINEAR PROGRAMMING: Models, Theory, and Computation*
Sethi, Yan & Zhang/ *INVENTORY AND SUPPLY CHAIN MANAGEMENT WITH FORECAST UPDATES*
Cox/ *QUANTITATIVE HEALTH RISK ANALYSIS METHODS: Modeling the Human Health Impacts of Antibiotics Used in Food Animals*
Ching & Ng/ *MARKOV CHAINS: Models, Algorithms and Applications*
Li & Sun/ *NONLINEAR INTEGER PROGRAMMING*
Kaliszewski/ *SOFT COMPUTING FOR COMPLEX MULTIPLE CRITERIA DECISION MAKING*
Bouyssou et al/ *EVALUATION AND DECISION MODELS WITH MULTIPLE CRITERIA: Stepping stones for the analyst*
Blecker & Friedrich/ *MASS CUSTOMIZATION: Challenges and Solutions*
Appa, Pitsoulis & Williams/ *HANDBOOK ON MODELLING FOR DISCRETE OPTIMIZATION*
Herrmann/ *HANDBOOK OF PRODUCTION SCHEDULING*
Axsäter/ *INVENTORY CONTROL, 2nd Ed.*
Hall/ *PATIENT FLOW: Reducing Delay in Healthcare Delivery*
Józefowska & Węglarz/ *PERSPECTIVES IN MODERN PROJECT SCHEDULING*
Tian & Zhang/ *VACATION QUEUEING MODELS: Theory and Applications*
Yan, Yin & Zhang/ *STOCHASTIC PROCESSES, OPTIMIZATION, AND CONTROL THEORY APPLICATIONS IN FINANCIAL ENGINEERING, QUEUEING NETWORKS, AND MANUFACTURING SYSTEMS*

* ***A list of the more recent publications in the series is at the front of the book*** *